Université de Montréal

Collaborative filtering techniques for drug discovery

par
Dumitru Erhan

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Août, 2006

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Collaborative filtering techniques for drug discovery

présenté par:

Dumitru Erhan

a été évalué par un jury composé des personnes suivantes:

Douglas Eck
président-rapporteur

Yoshua Bengio
directeur de recherche

Philippe Langlais
membre du jury

Mémoire accepté le

RÉSUMÉ

Cette thèse examine le problème d'apprendre plusieurs tâches simultanément, afin de transférer les connaissances apprises à une nouvelle tâche. Si on suppose que les tâches partagent une représentation et qu'il est possible de découvrir cette représentation efficacement, cela peut nous servir à construire un meilleur modèle de la nouvelle tâche. Il existe plusieurs variantes de cette méthode: transfert inductif, apprentissage multitâche, filtrage collaboratif, etc. Nous avons évalué plusieurs algorithmes d'apprentissage supervisé pour découvrir des représentations partagées parmi les tâches définies dans un problème de chimie computationelle. Nous avons formulé le problème dans un cadre d'apprentissage automatique, fait l'analogie avec les algorithmes standards de filtrage collaboratif et construit les hypothèses générales qui devraient être testées pour valider l'utilisation des algorithmes multitâche. Nous avons aussi évalué la performance des algorithmes d'apprentissage utilisés et démontrons qu'il est, en effet, possible de trouver une représentation partagée pour le problème considéré. Du point de vue théorique, notre apport est une modification d'un algorithme standard—les machines à vecteurs de support– qui produit des résultats comparables aux meilleurs algorithmes disponibles et qui utilise à fond les concepts de l'apprentissage multitâche. Du point de vue pratique, notre apport est l'utilisation de notre algorithme par les compagnies pharmaceutiques dans leur découverte de nouveaux médicaments.

Keywords: Apprentisage multitâche, Filtrage collaboratif, QSAR, Méthodes à noyaux, Réseaux de neurones

ABSTRACT

We investigate the problems of learning several tasks simultaneously in order to transfer the acquired knowledge to a new task. Assuming that the tasks share some representation that we can discover efficiently, such a scenario should lead to a better model of the new task, as compared to the model that is learned by only using the data for the new task. This technique has many names: inductive transfer, multi-task learning, learning to learn, collaborative filtering. All of these are varieties of the same idea that we try to exploit. We have evaluated several supervised learning algorithms in order to discover shared representations among the tasks defined in a computational chemistry/drug discovery problem. We have cast the problem from a statistical learning point of view, traced analogies with standard collaborative filtering techniques, and set up the general hypotheses that have to be tested in order to validate the multi-task learning approach. We have then evaluated the performance of the learning algorithms and showed that it is indeed possible to learn a shared representation of the tasks. From a theoretical point of view, our contribution also comprises a modification to the Support Vector Machine algorithm, which can produce state-of-the-art results using multi-task learning concepts at its core. From a practical point of view, our contribution is that this algorithm can be readily used by pharmaceutical companies for virtual screening campaigns.

Keywords: Multi-task learning, Content-based filtering, QSAR, Kernel Perceptron, SVM, Neural Networks

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

CF	Collaborative Filtering
HTS	High-Throughput Screening
MTL	Multi-Task Learning
MT-NNet	Multi-Task Neural Network
MT-SVM	Multi-Task Support Vector Machine
MT-JRank	Multi-Task JRank
NNet	Neural Network
QSAR	Quantitative Structure-Activity Relationships
ST-SVM	Single-Task Support Vector Machine
ST-NNet	Single-Task Neural Network
ST-JRank	Single-Task JRank
SVM	Support Vector Machine

NOTATION

General:

$\mathbb{R}$	The set of real numbers
$\mathbf{x} \in \mathbb{R}^{D_x}$	A d_x-dimensional vector in $\mathbb{R}$, which contains input features
$\mathbf{t} \in \mathbb{R}^{D_t}$	A d_t-dimensional vector in $\mathbb{R}$, which contains task features
y	A real number that is the the output corresponding to some pair $(\mathbf{x}, \mathbf{t})$
^{t}	Transpose of a vector/matrix
$K(\mathbf{x}, \mathbf{y})$	A kernel function evaluated for vectors $\mathbf{x}$ and $\mathbf{y}$
$\Psi(\mathbf{x}, \mathbf{t})$	Map of a pair of vectors $(\mathbf{x}, \mathbf{t})$ to a (high-dimensional) feature vector
$F(\mathbf{x}, \mathbf{t}; \mathbf{w})$	A linear transformation of $\Psi(\mathbf{x}, \mathbf{t})$, using $\mathbf{w}$
$f(\mathbf{x}, \mathbf{t}; \mathbf{w}, \mathbf{\Theta})$	A decision function for $(\mathbf{x}, \mathbf{t})$
$\alpha_{(\mathbf{x},\mathbf{t})}$	A learned parameter that corresponds to the pair $(\mathbf{x}, \mathbf{t})$

To my parents and to my girlfriend.

ACKNOWLEDGEMENTS

This thesis would have not been possible without the continuing support and guidance of my supervisor, Yoshua Bengio. My collaboration with Pierre-Jean L'Heureux was especially fruitful and enjoyable. Olivier Delalleau has been the person without whom many of the experiments described in this thesis would not have been possible. The time spent in the LISA and GAMME labs and the discussions with the guys and girls from there have certainly made the life more interesting throughout the whole master's program. Many thanks to all of you.

I would also like to thank Dr. Christopher Walpole and other AstraZeneca R&D Montreal staff for offering me the opportunity to work on this very interesting dataset. This work was supported financially in part by *Valorisation Recherche Québec* and the National Science and Engineering Research Council of Canada.

Last, but not least, I am very grateful to my parents and my girl-friend, who were of great help and provided lots of moral support during the last years!

CHAPTER 1

INTRODUCTION

The idea of learning more than one thing at a time is certainly not new. It is very likely that our brain does this constantly. We wanted to explore this idea in a more mathematical setting and apply our findings to a novel and very practical problem. This chapter introduces the reader to the general problem of learning and presents the setting and the motivation for our work on Multi-Task Learning or Collaborative Filtering for drug discovery. We describe the main contributions of this thesis and we guide the reader through the rest of the chapters.

1.1 Machine Learning

Machine Learning is the art of crafting techniques that allow computers to "learn". Generally speaking, it is a field of Artificial Intelligence that is closely related to the field of Statistics. Machine Learning (ML) is typically concerned with building algorithms for analyzing, deducing or inferring from data. Thus, Machine Learning research encompasses a broad spectrum of problems, such as the theoretical foundations of inference from data, the practical industrial considerations of learning algorithms, such as reducing their complexity, the analysis of the capacity of an algorithm to "learn" and to "generalize", the definitions of "learning" and "generalization", etc.

A most general and, at the same time, precise definition of "learning" is not in the scope of this thesis. However, it is intuitively clear that we can claim that, for instance, a decision-making algorithm is "learning" if the decisions of this algorithm, based on the data from which it has "learned" something, are at least better than randomness and at best as good as (or better than!) the ones made by a human that has access to the same data as the algorithm. In the long run, a "learning algorithm" should produce decisions that are better and better over

time, as more and more data is used for building these decisions.

An example of such a decision-making algorithm is one that distinguishes apples from oranges. A human that has seen neither would probably be able to distinguish these fruits after seeing one or at most a few examples of each. It seems that our brain is able to learn these kinds of things very quickly—perhaps by comparing and contrasting the different properties of these fruits. Properties such as the general shape (spherical or not), size, color, etc. could be used for constructing a "learning algorithm" that could use them in such a way so as to be able to *classify* a new example: either as an apple or as an orange. Presumably, the algorithm would combine the properties in a way that would reveal the connection between these properties and whether the given fruit is an apple or an orange. This algorithm would also presumably become better and better at classifying fruits as it will have more data to fine-tune the parameters that uncover the relationship between their properties and their identity.

This simple example introduces many of the concepts and problems that are frequently encountered in Machine Learning. The decision-making process of deciding which fruit is which is generally called *classification*. The phase of the algorithm that adjusts the parameters is called *training*. Choosing a set or a function of the parameters that performs best is *validation*. The process of verifying how well an algorithm works with a chosen set of parameters on new cases is called *testing*. The *generalization performance* of an algorithm given a set of parameters and data for training and validation can then be estimated during testing.

There are many aspects of Machine Learning that have been overlooked when describing this example; we have not touched on what and how many properties of objects should we consider, whether we can performe classification without knowing the *labels* (the identities) of the objects to be classified (by, for instance, *clustering* them), whether the number of such labels can be greater than one (*multi-class classification*) or even infinite (*regression*), whether learning is all about decision-making or not, etc. We need not consider these things for now—the big picture is that, more often than not, it is desirable to build algorithms that have a good

generalization performance.

Given such an algorithm, it would be interesting to know the answer to the following questions: is it possible to "re-use" the "knowledge", which one acquired by building this *model* of the data, for learning a new task? The human brain seems to do it on a regular basis and this is a plausible explanation for our ability to classify apples and oranges from very few training examples. Naturally, we would like a learning algorithm to do the same. It turns out that there exist methods for learning more than one task at a time and for transferring the acquired "knowledge" between the tasks that are learned. We discuss this in more detail in the next section.

1.2 Multi-Task learning and Collaborative Filtering

Multi-Task learning is known under a variety of names: learning to learn, inductive transfer, bias learning, collaborative filtering, etc. Each of these notions are small variations of the same idea—that one can construct learning techniques that can exploit acquired "knowledge" in order to bias the learning of a new task and improve the generalization performance. A more detailed treatment of the previous work in the field of Multi-Task Learning is presented in Chapter 2; for now, it suffices to say that the field was popularized in the 1990s by extending [15, 16] standard Neural Networks for learning multiple tasks at the same time.

Such work has shown theoretically and practically [8, 32] that taking into account multiple related tasks can be greatly beneficial to generalization, if the tasks are sufficiently related[1]. If the added tasks are unrelated, the generalization power could decrease, because spurious relations are learned, but there are cases when even unrelated tasks might be helpful [16].

One popular application of MTL are Collaborative Filtering [34, 60] systems.

[1] A necessary and sufficient condition for task relatedness is roughly the following: there exists a simpler—perhaps in the Kolmogorov complexity [49] sense—model that describes the joint distribution of inputs, outputs, and tasks, than the separate models of inputs and outputs that one would obtain for each task.

Such systems produce recommendations that are based on similarities between the preferences of different users of the system. Collaborative Filtering applications are usually online bookstores, movie rentals web-sites, online music shops, etc. where users of the system rate the products and where the system has to infer the ratings that a user would give to the items he/she has not rated yet. It is quite easy to draw the parallel between CF and MTL—predicting the preferences of one user is a single learning task, whereas modeling jointly the preferences of all the users of the system could be considered multi-task learning. As we will see later on, this observation enabled us to extend a collaborative filtering algorithm that we then applied to solving a particular multi-task learning problem that has very little in common with user preferences; in what follows, we present the practical motivation for our work.

1.3 Drug Discovery

The pharmaceutical industry is a multi-billion industry that relies heavily on new computer technology both in the process of drug development and in the process of finding drugs for new or established diseases. The drug discovery process can extend for several years and can cost short of a billion dollars for a single drug. It is thus quite desirable for a pharmaceutical company to apply methods for reducing the time and money spent on developing a new drug. Machine Learning techniques that help in building statistical models for evaluating potential drugs' likelihood of success is one of the computational approaches used in the industry.

One of the first stages of drug discovery is called High-Throughput Screening (HTS), during which a library of usually tens of thousands of compounds is tested against the target protein—which represents the "disease" that one tries to find a drug for—so that one can see how much the compounds influence this target. Based on these results, one will try to develop a better compound by finding quantitative structure-activity relationships (QSAR), i.e. correlations between the biological activity and the structure of the chemical compounds through statistical

means. These correlations enable the drug discoverer to model the link between the structure and the activity and find compounds whose structures correspond to a more desirable level of activity.

Even given the recent advances in robotics, the process of physically testing each compound is time-consuming and expensive. Combinatorial chemistry[2] techniques allow us to produce (virtually) millions of compounds. Statistically reliable and computationally feasible methods for performing "virtual screens" of these compounds are increasingly used in the pharmaceutical industry [13]. Virtual High-Throughput Screening is the process of building a model that "connects" the molecular features or structures (perhaps even the geometrical structures) of the compounds to their activity in the presence of a certain target.

Virtual HTS is in itself not an easy task, even if it usually does not involve a lot of laboratory work. This is because a reliable set of already tested compounds (the "training set") has to be present, the molecular features of these compounds have to be representative, and the statistical model has to be able to link these features to the activity in the presence of a certain target such that it can generalize well to previously unseen compounds. There has been a lot of work and success in applying Machine Learning methods to Virtual HTS problems. A round-up of these methods is presented in Section 2.3; for now, it suffices to say that the developments of the pharmaceutical research in this field follow very closely the developments in the Machine Learning community. This shows quite clearly that the pharmaceutical industry is always in need for new technologies that would enable companies to perform Virtual HTS campaigns more efficiently.

1.4 Multi-Target Virtual HTS

One interesting way of performing Virtual HTS more efficiently lies in exploiting the data from previous HTS campaigns. If these campaigns were performed on a set of related targets (we define such relatedness in Section 2.3) then it should be

[2]Fast synthesis of a large number of structurally related compounds

possible to transfer—in an inductive way, as described in Section 1.2—the knowledge acquired from the experiments to the virtual tests that are to be done on a new target (that is also related in some way to the targets for which we have experimental results). Such an algorithm could be put to good use by the pharmaceutical companies and in Section 2.3 we describe several scenarios in which such inductive transfer could help.

The parallel between collaborative filtering or multi-task learning and QSAR / Virtual HTS can be made almost immediately: the tasks (or the "users") are the biological targets, the inputs (or the "items") are the molecular compounds and the labels/outputs (or the "ratings") are the levels of activity of the given compound for a given target. The descriptors or the features of the targets and of the compounds could be anything that might help us in uncovering relationships both between the targets and the compounds.

1.5 Contributions of the Thesis

This work builds up on our article [27] and two poster presentations [26, 48] on the same topic and which cover the first parts of the thesis.

In this thesis, we investigate several questions related to the process of multi-task learning. First and foremost, we were interested in developing practical methods for measuring the degree to which we can profit from learning multiple tasks at the same time. Therefore, it is very interesting for us to see the evolution of the generalization of a multi-task learning algorithm when trained with only a small sample of data from a given task, as a function of the size of this sample. We are also interested in how it is possible to "transfer" the "knowledge" acquired by learning several tasks to a *completely new task*. Finally, we wanted to explore theoretical and practical ways of encoding the similarity or the relatedness of tasks and exploiting this for improving the multi-task learning algorithms that we used.

We have demonstrated that "pure" inductive transfer is possible in the context of a particular application, and that it can be quite helpful. We have defined a

clear way of testing, for a particular dataset, the degree to which multi-task learning helps, when compared to standard single-task learning. All the algorithms that we used have built-in ways of computing a similarity measure between tasks; one of these algorithms, the Multi-Task Support Vector Machine that uses a Collaborative Filtering-inspired kernel matches the state-of-the-art performance and is a clear candidate for inclusion into the industrial process of drug discovery.

From the point of view of the drug discovery process, our objective and contribution was to compare and evaluate methods to take advantage of the commonalities between the different targets within a target class. In addition, we developed a solution that allows us to estimate QSAR models for so-called "orphan targets" that have not yet been tested, or for which there are very little available data. The goal of our approach was not to create the best global predictive model for a collection of accurately known targets. We assumed that we do not know the structure of the targets, because we want to generalize to a new unknown target. We have thus developed a practical approach where very little prior knowledge of the target is needed; we were less interested in building the best model for a single target than building a model for which we lack sufficient data. Finally, to the best of our knowledge, such a (multi-target) dataset has never been discussed before in the computational chemistry literature. This thesis (along with the afore-mentioned journal paper and presentations) is therefore the first to offer insights into this kind of dataset and ways to solve problems defined by it.

To summarize: from the theoretical point of view, our contribution is the analysis of Multi-Task Learning when one of the tasks is either completely unknown to the learning algorithm or for which we have only a small training set. Practically speaking, we tested a well-known Multi-Task learning algorithm (MT-NNet) and modified a published algorithm (MT-JRank) to produce the Multi-Task Support Vector Machines which achieves state-of-the-art results for a novel drug discovery problem. This algorithm is also enabling us to conclude that, for the given drug discovery dataset, "pure inductive transfer" is possible, which is a very promising result.

1.6 Structure of the Thesis

This thesis starts with an overview of the work done in the field of Multi-Task Learning/Collaborative Filtering. Sections 2.1 and 2.2 discuss the developments in this field, from the first Neural Network-based models to the more modern kernel-based approaches, and contain an overview of the theoretical insights behind Multi-Task Learning. Section 2.3 contains a short listing of the methods that have been used for solving the Virtual HTS problem. Finally, Section 2.4 draws the parallels between the research presented in the previous sections and this thesis.

We then proceed to formally describe the problem to be solved in Section 3.1. Sections 3.2, 3.4, 3.5 present the techniques that we used: a Multi-Task Neural Network, a kernel perceptron-based Collaborative Filtering algorithm called JRank, and a Multi-Task Support Vector Machine. In Section 4.1 we discuss the experimental setup that is common to all the algorithms, whereas in Section 4.2 we present the experimental results obtained with each of the techniques.

Chapter 5 is a discussion of the results. We analyze possible extensions of our techniques and future directions of work in Section 5.2 and we conclude the thesis with Chapter 6, which summarizes the work done and the results obtained.

CHAPTER 2

PREVIOUS WORK

In this chapter, we will go through the main developments of multi-task learning and collaborative filtering, analyze the main ways that Machine Learning techniques are used in the computational chemistry/drug discomulvery community, suggest ways of applying multi-task learning/collaborative filtering techniques to solving the drug discovery problem using multiple related biological targets and trace parallels from our approaches to the previous work done in the field.

2.1 Multi-task Learning

One of the first attempts at constructing an efficient procedure for learning more than one task at a time was by extending standard multi-layer neural networks [62]. Neural networks provided an ideal testbed for implementing multi-task intuitions: there existed an efficient algorithm for training them, the translation of intuitions into concrete models was relatively easy and the multi-task models were computationally not much more expensive than simple, single-task learning models.

The simplest of such extensions was to create a shared hidden layer that is trained in parallel for all the learning tasks. Figure 2.1 (taken from [16]) presents such an extension. In this case, the training procedure would be done on all the tasks in parallel and because the structure of the network includes a shared layer (weight matrix), it is possible for so-called "shared internal representations" to develop and to be learned. There are other architectures possible, but most, if not all of them are varieties of the same idea: that the tasks share some connections in the neural network. Caruana [16] provides a host of examples of such networks, and convincing results that show that such networks can indeed learn several tasks at the same time, better than equivalent single-task learning networks.

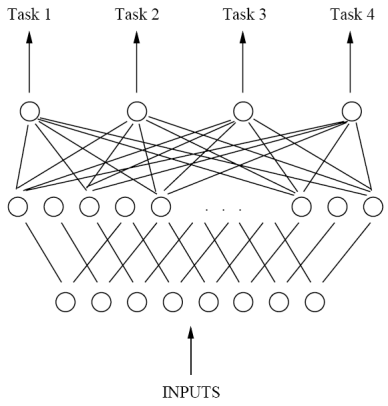


Figure 2.1: A simple extension to the standard single-layer neural network architecture, which allows for multiple tasks to be learned at the same time, thus creating a so-called "shared internal representation"

A parallel development [7] (but in a Bayesian framework) introduced the notion of an *objective prior distribution*, from which a learner samples the related tasks that are to be learned. This *environment* that contains the sampled tasks provides the multiple datasets that correspond to these tasks. Given this learner and a way to sample from such an environment, the learner can then search for the hypothesis that best explains the tasks. The same paper gave bounds on the information needed to learn a task, when it is learned concurrently with other tasks. Baxter [9] then expanded on that and gave bounds on the number of tasks that are sufficient in order to learn a novel task. These results lay the theoretical foundations for multi-task learning and generalized the insights gained from extended neural networks to handle multiple tasks.

Bakker [2] has expanded on the intuitions behind the multi-task neural networks and behind the theoretical results of Baxter and has introduced a hierarchical Bayes model that can also perform clustering of tasks. This is done by setting the prior over the shared parameters in a multi-task neural network as a mixture of Gaussians. The model can also account for more fine-grained relationships between tasks. This is obtained by introducing task-specific features and setting the first

moments of the priors as a function of these features.

Ben-David and Schuller [10] demonstrated a set of simple transformations that defined *task relatedness*. Their approach is based on comparing the distributions that generate the data for the tasks and using the similarity between these distributions to present bounds on how much a learning algorithm can profit from performing multi-task learning. Their approach is interesting as it provides a way of quantifying (albeit by a very simple measure) the relatedness of tasks.

The task relatedness idea can be viewed from a slightly different angle in the context of kernel machines [70]. Evgeniou, Micchelli, and Pontil [28,29] generalized the popular Support Vector Machine algorithm [65] to use similarity measures and objective functions that take into account multiple tasks. They achieve this by using a regularized functional that couples the tasks and provides for a very explicit way of specifying the type of relatedness between tasks. Their approach can also accommodate for non-linear kernels. Their approach is also one of quite a general way of explicitly stating the relationships between tasks.

Multitask learning has been also been applied in a variety of settings. Predicting pneumonia mortality is a popular example [19], but fields as varied as sensor fusion for robotics [21], stock selection [33] and lifelong learning [67] have also profited from the algorithms developed in this field.

Most of the techniques that we just described attempt to use multi-task learning in order to improve the generalization performance of a task that encompasses them all (multi-task learning is viewed as learning a "common goal", in a sense). As laid out in the introduction, our goal is to find a way to transfer the knowledge acquired by performing multi-task learning to a new task, that does not "encompass" the rest of the tasks, but that is simply similar to them, under some similarity measure. While some theoretical foundations for doing that in a probabilistic setting have been presented before [68], to the best of our knowledge, there are little to no experiments that have been performed for generalizing to a completely new and unseen task. In this thesis, we will present experimental results for this scenario.

The techniques that we have presented so far are quite limited, computationally

speaking, in the number of tasks that one can learn with them. They can rarely accommodate for more than several hundred tasks. Imagine, however, that we are in a collaborative filtering scenario, where we are presented with several hundred thousand preference profiles of users, which are our learning tasks. Some of these techniques, specifically the kernel machines, would be computationally too expensive to use in such a scenario. In the following section, we present several ideas that make it possible to learn efficiently in such settings.

2.2 Collaborative and Content-Based Filtering

Collaborative filtering [34, 60] has its roots in recommender systems applications, whereby automated recommendations are produced. Such recommendations are based on similarities between the preferences of different users of the system. Typical collaborative filtering datasets usually include some form of demographic data about the users of the system and/or some basic facts about the items (movies, songs, etc.) that are rated. Evidently, such data could be useful in improving the generalization performance of the algorithm, especially when for some user or item there is only a small number of ratings available. Systems that make use of such extra data have been termed *content-based filtering* [4] algorithms.

Breese et al [14] contains an overview of the basic techniques that are used in the collaborative filtering community. That paper identified are two main categories of such techniques. The first typically treats the ratings that users gave to items as a big sparse matrix and attempts to fill the missing values by applying a fixed function that is dependent on the observed ratings. Another similar approach is to perform a Singular Value Decomposition of the ratings matrix and fill the missing values based on this decomposition [43]. This techniques are essentially non-parametric methods for learning in a collaborative filtering setting.

The second category, which is the one we are more interested in, is concerned with *modeling* the missing values in the ratings matrix. Practically all the main machine learning techniques have been used for this purpose. Probabilistic ap-

proaches, such as the Probabilistic Latent Semantic Analysis [40,41], are a popular alternative to the decomposition-based techniques, partly because of the usual (real-world) assumption that they make, which is that users are assumed to be characterized by a certain profile that they belong to (most of these are a clustering schemes, essentially). Similar techniques perform simultaneous hard or soft [69] clustering of users and items. Probabilistic extensions of both of these approaches exist, too [53].

Other probabilistic approaches that have been used for CF are Bayesian networks [14, 57], dependency networks [39] and Gaussian processes [18]. Decision trees [14] and boosting [30] are also among the popular choices for this application.

There have been many attempts at incorporating item-specific features [63] into the learning procedure (content-based filtering), but very few of these have also incorporated user-specific features (age, sex, location, etc.). Ideally, one is interested in using all the data that is available—both ratings and user/item descriptors—such that the algorithm could exploit to the maximum the relationships between the users, items and the ratings. Such an algorithm would be a combination of collaborative and content-based filtering. Basu et al. [6] made use of this idea for the first time, but the user-specific features that they used were actually inferred from the ratings that users gave and the features of the items that they gave ratings to. Obviously, these user features do not add more actual information to the learning procedure. Basilico and Hofmann [40] built up on the idea and presented an algorithm that can make use of arbitrary real-valued features and fairly general similarity measures.

Their approach makes use of the same intuitions as most of the multi-task learning approaches. They consider the similarity (relatedness) between users and quantify this relatedness through some concrete user features and a concrete distance measure between users having these features. In the earlier collaborative filtering approaches the "distance" between users was proportional to the correlation between the ratings that they gave to the same items. The parallel with the earlier multi-task learning approaches can be made here as well—there the "dis-

tance" between tasks was a function of the "correlation" between the inputs and outputs of the tasks.

Such notions naturally gave rise to the following question: would it be possible to build a model that can give an estimate of the preferences of a user, if the only thing that we know about the user are demographic data (i.e. the user has not rated any item in the system). This is typically referred to as the "cold start" problem in collaborative filtering research and has received some attention, albeit limited one. There is a clear parallel between this problem and the problem of generalizing a multi-task algorithm to a completely new and unseen task, for which we only know some task-specific features.

One of the first ways of approaching the cold start problem is presented in [64]. The idea is to assign probabilistically each user into a cluster, based on the user features, and to estimate the preferences of a new user based on his features and the cluster that he would be assigned to. While the paper does present several useful metrics for comparing algorithms in such a scenario, the results are not encouraging, since the method does not perform much better than a simple baseline. While there have been other attempts, none of them shows a convincing way of solving the cold start problem. This thesis is also an attempt at solving it, except that we posit the problem in a computational chemistry setting.

The approach presented by Basilico and Hofmann [5], of using a kernel to measure similarity between user-item pairs, is generalized by Evgeniou and Pontil [28], in the sense that they present a principled way of viewing multi-task learning problems as convex optimization problems. They describe several fairly general techniques for doing that, one of them using task descriptors (user features, in the collaborative filtering context). Our approach is also an extension of [5], except that it is less general.

2.3 Virtual High-Throughput Screening/QSAR

Virtual High-Throughput Screening (Virtual HTS) / Quantitative structure-activity relationship (QSAR) emerged as a valuable technique for the pharmaceutical sciences some thirty years ago [38,66]. It has since evolved into many branches of research (review in [46]) and surfed on the growing capabilities of computers, like the development of neural networks [73] and 3D-QSAR [47].

Neural networks have been especially popular [1, 23], due to their ubiquitousness and the ease with which one could translate the intuitions behind a QSAR model into an actual algorithmic model. Genetic algorithms [24] and decision trees [45] have also been used with varying degrees of success. In recent years, several other groups have introduced kernel machines [54] and Support Vector Machines in QSAR [55, 71]. These techniques have often proved superior to Partial Least Squares or neural networks, the more traditionally used algorithms.

Ensemble methods, such as boosting and bagging have also gained in popularity [52], partly because of studies that showed their edge over single learning algorithms. We have chosen two main classes of algorithms for our comparison: one is a traditional neural network and the others are the kernel machines, which are considered to be state-of-the-art in this domain [55].

In Section 1.3 we described the general scenario in which multi-task learning could be helpful for in the context of drug discovery research. We have stated that there could exist biological targets that are related and for which we have screening data. Interestingly, such multiple related targets do exist in the pharmaceutical industry, where they are commonly called a *target class*, e.g., kinases, G-protein coupled receptors, etc. These target classes have some common features. First, they represent some significant portion of a therapeutic area (in our case, the targets are related to the area of relieving "pain"). Some members of these target classes have been well studied. Second, targets within each of these target classes share a common structural frame. Members of each target class may have a similar binding

site[1]. Third, with the development of genomic projects, many new members of these target classes have been identified, though the biological roles of these newcomers (so called orphans) are still unknown. The challenge we are facing here is how to transfer our knowledge from known targets to orphans. As mentioned above, the traditional statistical approach (Virtual HTS) considers a different machine learning task for each member of a given class. We would like to extend that with the concepts from Multi-Task Learning.

In Section 4.1.1 we describe in detail the dataset that we are using. This dataset has all the characteristics of a collaborative filtering dataset: the targets have features, the chemical compounds have features as well, and the matrix that describes the interactions between the targets and the compounds is sparse. All the techniques that we employed are inspired by (or taken directly from) research on collaborative and content-based filtering, hence the title of this thesis.

2.4 Parallels to the Thesis

Our work builds on the ideas from the above: we are interested in measuring task relatedness, in learning a completely new task, in using task-specific features to encode task relatedness and in devising techniques for improving the generalization performance while using multi-task learning for a specific computational chemistry dataset. Such an application of multi-task learning techniques to such a dataset has never been done, to the best of our knowledge. However, the techniques that we employed are simple extensions to those that are popular in the drug discovery industry and research [13].

More specifically, we measure task relatedness as a function of the generalization performance of the multi-task learning algorithm, as in [15]. We use task-specific features as done by [5] and [28]. One of our approaches is also an improvement over [5] and is quite similar to [28], except that the objective function that our multi-task support vector machine algorithm minimizes is slightly different. The neural

[1]The region on the target to which specific compounds form chemical bonds.

network architecture that we employed is very similar to the types of architectures employed by [15], except that we use task-specific descriptors. Our approach is novel from the CF point of view, as it tackles the problem of "cold starting" and provides a way of overcoming it.

As one can see, we build up on previous work and our improvements are relatively incremental. However, we have carried very extensive experiments with an important and very large dataset. We do not see the algorithmic part as the main contribution of this thesis. The insights into the problem and the dataset, which these algorithms have provided us with, are, in our opinion, the more important part of this work.

CHAPTER 3

PROPOSED ALGORITHMS

In this chapter, we describe in detail the proposed algorithms for transferring knowledge acquired from learning several tasks collectively to a new task. Each section contains pseudo-code, run-time analysis, and considerations that have to be taken care of in order to solve the above problem using the QSAR dataset.

3.1 Formal Definition

Before proceeding to the description of the algorithms, we would like to provide the reader with a formal definition of the problem.

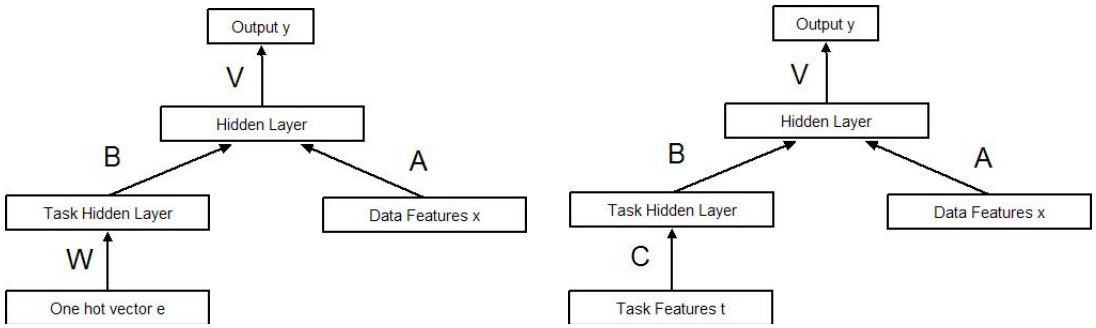
Generally speaking, assume a collection of k datasets, where each dataset corresponds to a (classification, regression, etc.) task that is to be solved. Each of these datasets consists of n_k pairs $(\mathbf{x}_i^k, y_i^k), i = 1 \dots n_k$, where $\mathbf{x}_i^k \in \mathbb{R}^{D_x}$ and $y_i^k \in \mathbb{R}$ (the $\mathbf{x}_i^k$ can and will overlap across the datasets and are assumed to be from the same underlying space). Assume that for each of these datasets we are also given a vector $\mathbf{t}^k \in \mathbb{R}^{D_t}$, which is a set of task-specific descriptors or features. We are interested in finding algorithms that would be able to exploit this data in such a way such that when presented with a new dataset of n_{new} pairs $(\mathbf{x}_j^{new}, y_j^{new}), j = 1 \dots n_{new}$ and a vector $\mathbf{t}^{new} \in \mathbb{R}^{D_t}$, they would be able to generalize well to this dataset, i.e. predict y_j^{new} well, according to some loss functional, given x_j^{new} and $\mathbf{t}^{new}$. The algorithm must generalize well *without having seen* any of the $(\mathbf{x}_j^{new}, y_j^{new}, \mathbf{t}^{new})$ triplets or after seeing a very small sample of the triplets from the new task.

In our computational chemistry/drug discovery case, the triplet $(\mathbf{x}_i^k, y_i^k, \mathbf{t}^k), i = 1 \dots n_k$ corresponds to the event of testing molecule with descriptor $\mathbf{x}_i^k$ on target k, having target descriptor $\mathbf{t}^k$ and with the result of the test being y_i^k ($y_i^k = 1$ means that the compound was active in the presence of the target, $y_i^k = 0$ means that it was inactive). As mentioned above, we assume that the molecule descriptors are

sampled from the same underlying space. We posit this assumption for the target descriptors, too.

3.2 Multi-Task Neural Network

As mentioned in the introduction, the first techniques for modeling more than one task at the same time were developed in the context of multi-layer neural networks, which were modified so as to allow the process of *inductive transfer* (from one modeling task to another) to take place. We have developed a neural network architecture that is based on such ideas. The basic architecture of the neural network model, shown in Figures 3.1a and 3.1b, has two hidden layers. The first hidden layer is committed to processing task descriptors, in order to discover relationships between the tasks. The architecture assumes that such relations can be uncovered more easily when we perform a low-dimensional embedding for task descriptors.



(a) No task features (b) Using task features

Figure 3.1: Multi-task Neural Network architectures

In one version, shown in 3.1a and in equation 3.1, we use as the input of the first layer a one-hot variable (a vector $\mathbf{e}_k$ full of zeros except for a 1 at position k for coding symbol k) indicating the task number. The second layer receives the output of the first layer and the descriptors of the input $\mathbf{x}$. Note that such an architecture does not use any task descriptors except for an indicator of which task a specific

input vector "belongs" to. Therefore, this architecture will learn an individual predictive model for each task, but the first layer will contain information about the relatedness of tasks with respect to the correlations in the mapping between input vectors and outputs (as this is the only way of learning any relatedness between tasks, in the absence of any other information). The precise mathematics for this model are:

$$P(y_i^k = 1|\mathbf{x}_i^k, k) = \text{sigmoid}(\mathbf{V}\tanh(\mathbf{A}\,\mathbf{x}_i^k + \mathbf{B}\tanh(\mathbf{W}\,\mathbf{e}_k)) \tag{3.1}$$

where $\mathbf{e}_k$ is the one-hot variable defined above. The learned weights matrix $\mathbf{W}$ will contain the low-dimensional embeddings for each task and will, in a sense, summarize the relationships between the targets (their "position" in this low-dimensional space). In another version, shown in 3.1b and in equation 3.2, we use task descriptors in the first layer as an aid for finding the relatedness of tasks. Here we learn an indirect predictive model for each task and the "positions" of the tasks in the low-dimensional embedding should be approximated better, given task descriptors that allow the algorithm to do so.

$$P(y_i^k = 1|\mathbf{x}_i^k, \mathbf{t}^k) = \text{sigmoid}(\mathbf{V}\tanh(\mathbf{A}\mathbf{x}_i^k + \mathbf{B}\tanh(\mathbf{C}\,\mathbf{t}^k))) \tag{3.2}$$

The parameters of the neural network ($\mathbf{V}$, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{W}$) or ($\mathbf{V}$, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$) will be tuned by stochastic gradient ascent [12, 62] on the average log-likelihood of the training set (average of the logarithm of the above probabilities). Details of the learning procedure can be found in Section 4.1.4.

Among the reasons for choosing this algorithm are the easiness of interpretation of the models, the speed of training and testing and its ubiquitousness in the computational chemistry community. We also felt that it would provide for a good baseline.

The algorithm can be applied to our computational chemistry setting as is, without further modifications. The task descriptors correspond to the biological target descriptors and the input vectors are the molecular compounds features.

3.3 An input-task similarity measure

We have mentioned a couple of times the notion of similarity between tasks (through their "position" in a low-dimensional space, for instance). It would be interesting to formalize this notion and to put it to use in a framework that would learn relationships between the inputs+task descriptors and the outputs in a statistically consistent way.

One way of doing that is through the use of kernels. These are nothing but predefined similarity measures which, under fairly general assumptions, can be used to train very efficient learning algorithms that discriminate between patterns. It is easy to see that we have two types of similarities that we can use in our setting: a similarity measure between input vectors (molecular compounds) and one between tasks (biological protein targets). A learning algorithm that generalizes across tasks has to somehow combine the two similarities in order to compute a more general measure of similarity, that between *input-task pairs*. In the case of MT-NNet, this is done in the second hidden layer, where the shared representation of tasks (their representation in the low-dimensional embedding) is combined with the representation of the input features.

This measure of similarity, the *input-task kernel*, can be of the type used in Support Vector Machine algorithms [65]—i.e. it can be a non-linear function of inputs and task descriptors and can project these into a high-dimensional space. Typically, algorithms that learn relationships between input-task pairs and outputs using this kernel measure will find a separating hyperplane in the resulting high-dimensional space. This hyperplane will separate in some non-linear way in the input space the examples from the two classes.

Let us formalize the intuitions behind the idea of an input-task similarity measure. We try to find a map Ψ that takes pairs $(\mathbf{x}, \mathbf{t})$ into $\Psi(\mathbf{x}, \mathbf{t}) \in \mathbb{R}^D$, where $\mathbf{t}$ is the vector of task features and $\mathbf{x}$ is the vector of inputs, for a given input-task pair (with D being the—possibly infinite—dimension of the resulting combined space). Such a map would allow us to compute similarities between *pairs* of inputs/tasks

and would allow us to generalize across both task features and input features at the same time.

Let $\mathcal{T}$ be the set of tasks, $\mathcal{I}$ be the set of inputs and the map be $\Psi : \mathcal{I} \times \mathcal{T} \rightarrow \mathbb{R}^D$, which gives a D-dimensional feature vector for each input-tasks pair. Our goal is then to choose a function, which should be optimal in some sense, from the set of functions F, which are linear in Ψ, i.e.,

$$F(\mathbf{x}, \mathbf{t}; \mathbf{w}) = \Psi(\mathbf{x}, \mathbf{t})^t \mathbf{w} \tag{3.3}$$

(where ^{t} is the transpose). This function would encode (in a linear fashion) the relationship between the input-task pair features and, combined with the respective outputs, will be tuned to fit some optimality criteria on the training set.

Note that $\Psi(\mathbf{x}, \mathbf{t})$ from equation 3.3 is not computed directly (for reasons that will become clearer shortly) and that our algorithm is only using dot-products in the feature space defined by Ψ. The dot product between the application of Ψ on two pairs is referred to as a *kernel*. More precisely, for two given pairs $(\mathbf{x}_i^k, \mathbf{t}^k)$ and $(\mathbf{x}_j^m, \mathbf{t}^m)$, we define the kernel as $K((\mathbf{x}_i^k, \mathbf{t}^k), (\mathbf{x}_j^m, \mathbf{t}^m))$ and it is a function that computes the similarity between these pairs. We will see shortly how to compute this measure efficiently.

As shown by Crammer and Singer [20] and Schölkopf and Smola [65], one can rewrite equation 3.3 as follows, thanks to the Representer Theorem:

$$F(\mathbf{x}_i^k, \mathbf{t}^k; \alpha) = \sum_{(\mathbf{x}_j^m, \mathbf{t}^m)} \alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)} K((\mathbf{x}_i^k, \mathbf{t}^k), (\mathbf{x}_j^m, \mathbf{t}^m)) \tag{3.4}$$

where $\alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)}$ is a vector of coefficients for each input-task pair from the training set. The way to compute these coefficients such that they minimize some loss functional is what sets apart different learning algorithms that use kernels. Thus if we can evaluate efficiently the kernel, then we do not need to explicitly compute the feature vectors given by Ψ. This is important because the computation of Ψ may be impractical if we want this non-linear transformation to be rich enough: in

practice we choose not Ψ but the kernel K, and for many choices of interest for K, the corresponding Ψ is infinite-dimensional. The only constraint on the choice of K is that it must be positive semi-definite[1].

Now we must define this general similarity measure. We take a bottom-up approach, by first defining similarity measures between pairs of tasks, then between pairs of inputs, and then combining the two measures into a kernel function of the desired type. Thus, we can use the following (non-exhaustive) list of kernels:

1. an identity kernel $\mathcal{K}_{\mathcal{T}}^{id}$, which returns one if the two tasks have the same feature vector and zero otherwise (this forces the Gram matrix to be of the required type),

2. a Gaussian kernel $\mathcal{K}_{\mathcal{T}}^{ga}(\mathbf{t}^k, \mathbf{t}^m) = \exp\left(-\frac{||\mathbf{t}^k - \mathbf{t}^m||}{2\sigma^2}\right)$, with σ^2 being a tunable hyper-parameter.

3. a correlation kernel $\mathcal{K}_{\mathcal{T}}^{co}$, which computes the Pearson correlation coefficient, which is a dot-product between the normalized output vectors ($\mathbf{y}^k$ and $\mathbf{y}^m$) corresponding to each tasks (over the inputs that are shared by the two targets). The Gram matrix corresponding to this similarity measure is not however positive semi-definite, and one way of making it positive semi-definite is by defining the following kernel:

4. a quadratic kernel $\mathcal{K}_{\mathcal{T}}^{qu}$, which is $\mathcal{K}_{\mathcal{T}}^{co} \cdot \mathcal{K}_{\mathcal{T}}^{co}$ (it has the necessary property of always being positive semi-definite).

We can define in a similar way $\mathcal{K}_{\mathcal{I}}^{id}, \mathcal{K}_{\mathcal{I}}^{ga}$, and $\mathcal{K}_{\mathcal{I}}^{qu}$, the kernels for the input features. So far, we have not mentioned a way of combining the kernels. If we were to deal only with task features (or only with the input features), combining the kernels could be done by simple addition, possibly also via a weighted sum, since

[1]It means that for any finite set $\mathcal{P}$ of input-task pairs s_i, the Gram matrix G associated with $\mathcal{P}$ must not have any negative eigenvalues. The entry (i, j) of G is $G_{ij} = K(s_i, s_j)$ with $s_i \in \mathcal{P}$ and $s_j \in \mathcal{P}$.

the weighted sum of positive semi-definite matrices is also positive semi-definite:

$$\mathcal{K}_{\mathcal{T}} = \mathcal{K}_{\mathcal{T}}^{id} + \mathcal{K}_{\mathcal{T}}^{ga} + \mathcal{K}_{\mathcal{T}}^{qu}. \tag{3.5}$$

We can do exactly the same for the kernel of the input features:

$$\mathcal{K}_{\mathcal{I}} = \mathcal{K}_{\mathcal{I}}^{id} + \mathcal{K}_{\mathcal{I}}^{ga} + \mathcal{K}_{\mathcal{I}}^{qu}. \tag{3.6}$$

If we are interested in combining $\mathcal{K}_{\mathcal{T}}$ and $\mathcal{K}_{\mathcal{I}}$, so that we can compute the similarity between input-task pairs, we could use the tensor product to get $K((\mathbf{x}_i^k, \mathbf{t}^k), (\mathbf{x}_j^m, \mathbf{t}^m))$. Intuitively, two given pairs should be most similar if and only if $\mathcal{K}_{\mathcal{T}}(\mathbf{t}^k, \mathbf{t}^m)$ is at its maximum and $\mathcal{K}_{\mathcal{I}}(\mathbf{x}_i^k, \mathbf{x}_j^m)$ is at its maximum, too. We cannot for any practical purpose, compute the tensor product (because of the infinite dimension vectors), but it turns out that the product is equivalent [65] to

$$K((\mathbf{x}_i^k, \mathbf{t}^k), (\mathbf{x}_j^m, \mathbf{t}^m)) = \mathcal{K}_{\mathcal{T}}(\mathbf{t}^k, \mathbf{t}^m) \cdot \mathcal{K}_{\mathcal{I}}(\mathbf{x}_i^k, \mathbf{x}_j^m) \tag{3.7}$$

which is a handy shortcut that also follows our intuitions!

Given this kernel and the definition of the F function that is to be learned (from equation 3.4), we can now move on to defining learning algorithms that could use these and the outputs y_i^k in order to learn a way to discriminate between input-task pairs.

3.4 JRank

The first such algorithm that we will consider is called JRank. It was proposed by Basilico and Hofmann [5] and it was the first to use the above idea of unifying task and input features into a common framework, albeit in a different problem setting, where the tasks corresponded to people and inputs corresponds to items that people rated (so a combination of content-based and collaborative filtering).

The underlying structure of the algorithm is very similar to the original percep-

tron [61], which means that it has several useful characteristics such as its simplicity and its online nature. It is a kernel-based extension of the original perceptron algorithm; such an extension is typically referred to as the *kernel perceptron* [31].

The essence of the algorithm is as follows. We are interested in performing *ordinal regression*, that is we would be interested in learning an *ordinal value* for each input. This contrasts to the more common regression and classification problems in that the numerical value of the output is not important. What is important is the *order* that we define on the outputs.

In order to represent this intuition in a mathematical way, the output of the function F is binned via a set of adaptive thresholds $\boldsymbol{\theta} \in \mathbb{R}^k$, where k is the number of "output levels" (ordinal values) we are interested in ($\boldsymbol{\theta}_k = +\infty$ for convenience). This is done in order to predict the output level from an input-task pair: by simply selecting the number of the bin where $F(\mathbf{x}, \mathbf{t}; \mathbf{w})$ falls into. The prediction function $f(\mathbf{x}, \mathbf{t}; \mathbf{w}, \boldsymbol{\theta})$ depends straightforwardly on $\boldsymbol{\theta}$: it outputs a level i associated with the interval $[\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1})$ which contains $F(\mathbf{x}, \mathbf{t}; \mathbf{w})$.

On a more fundamental level, what JRank is doing is finding a set of k hyperplanes in the feature space defined by Ψ. The space defined by two adjoining hyperplanes corresponds to a given "output level" (ordinal value). JRank will find this set by moving along the gradient of the loss functional and finding a local optimum.

The framework of ordinal regression is appropriate for both binary classification problems—where we would interpret the two output levels as "high" and "low"—and for multi-class / regression problems, where the transformation of the numerical values to an ordinal scale poses no problem.

Algorithm 1, as described in [5], is a straightforward extension to the kernel perceptron algorithm [31]. As in [20], JRank projects each instance from our dataset onto the real line. Each ranking is then associated with a distinct sub-interval of the reals. During learning these sub-intervals are updated: if and when the current set of parameters predicts an incorrect sub-interval, the parameters are updated such that the new predicted rank is closer to the sub-interval (and vice-versa, by

Algorithm 1 JRank

{α is a sparse parameter matrix, one element per experimental observation}
{$A_{(\mathbf{x},\mathbf{t})}$ is the output level corresponding to input $\mathbf{x}$ and task $\mathbf{t}$}
$\alpha_{(\mathbf{x},\mathbf{t})} = 0, \forall A_{(\mathbf{x},\mathbf{t})}$
{$\boldsymbol{\theta}$ is a vector of thresholds, defining the bins for the ordinal values}
$\boldsymbol{\theta}_i = 0, \forall i = 1, \ldots, k-1$ and $\boldsymbol{\theta}_k = +\infty$
{N_{it} is the number of iterations}
for $n = 1$ to N_{it} **do**
  for all $A_{(\mathbf{x},\mathbf{t})}$ **do**
    {The estimated activity level (equation 3.3)}
    $\hat{a} = f(\mathbf{x}, \mathbf{t}; \alpha, \boldsymbol{\theta})$
    {If the estimated activity level is incorrect, we update the parameters}
    if $\hat{a} > A_{(\mathbf{x},\mathbf{t})}$ **then**
      {Following the gradient}
      $\alpha_{(\mathbf{x},\mathbf{t})} = \alpha_{(\mathbf{x},\mathbf{t})} + \hat{a} - A_{(\mathbf{x},\mathbf{t})}$
      {The value of the F function becomes closer to the correct bin}
      for $i = A_{(\mathbf{x},\mathbf{t})}$ to $\hat{a} - 1$ **do**
        $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i + 1$
      end for
    else if $\hat{a} < A_{(\mathbf{x},\mathbf{t})}$ **then**
      {Following the gradient}
      $\alpha_{(\mathbf{x},\mathbf{t})} = \alpha_{(\mathbf{x},\mathbf{t})} + \hat{a} - A_{(\mathbf{x},\mathbf{t})}$
      {The value of the F function becomes closer to the correct bin}
      for $i = \hat{a}$ to $A_{(\mathbf{x},\mathbf{t})} - 1$ **do**
        $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - 1$
      end for
    end if
  end for
end for

modifying the boundaries of the sub-intervals).

$A_{(\mathbf{x},\mathbf{t})}$ is the output level observed for the pair $(\mathbf{x}, \mathbf{t})$ (the y_i^ks, essentially). In the formulation it is also understood that we have access to the set of all the data triplets $(\mathbf{x}, \mathbf{t}, A_{(\mathbf{x},\mathbf{t})})$. Before learning, the sparse parameter matrix α has non-zero entries $\alpha_{(\mathbf{x},\mathbf{t})}$ only for the observed pairs $(\mathbf{x}, \mathbf{t})$. It can be used for prediction via equation 3.4. A set of thresholds/bins $\boldsymbol{\theta}_i$ (one per ordinal value) is also learned. The algorithm runs through the dataset in N_{it} stages/iterations, updates $\alpha_{(\mathbf{x},\mathbf{t})}$ and updates the thresholds if it predicts an incorrect activity level. The algorithm assumes that the ranks are ordered from $i = 1$ to k, but it can be easily modified to accommodate other types of ranks.

The updates of $\alpha_{(\mathbf{x},\mathbf{t})}$ follow the prediction error (the difference between the predicted rank and the actual rank, also called the *ranking loss*), i.e., they follow the gradient, while the thresholds $\boldsymbol{\theta}_i$ are updated so that the value of the F function becomes closer to the correct bin at the next iteration. This is illustrated in Figure 3.2 (taken from [20]).

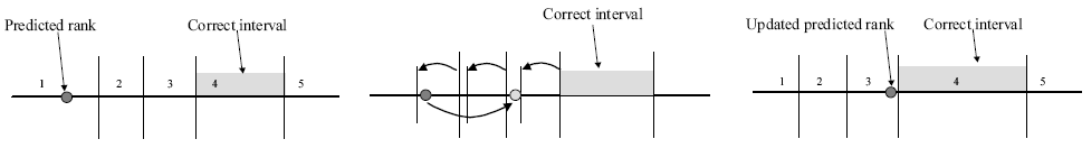


Figure 3.2: Intuition behind the update rule of JRank

The algorithm has two hyper-parameters:

1. The width of the Gaussian kernel σ. Ideally, there should be one for each (task and input) kernel.

2. The number of iterations N_{it}.

It is worth noting that the algorithm functions correctly and as expected when $k = 2$, i.e., it learns to perform binary classification. The algorithm reduces to

simple single-task learning if the dataset has only one target (α becomes a vector)—which is quite handy because it allows us to compare directly single-task learning (ST-JRank) with multi-task learning (MT-JRank).

In its most general form, the algorithm needs output levels in order to learn. This is a desirable feature, since very often in a computational chemistry context we are interested in learning different *activity levels*. Thus given a compound and a target, the compound can be "inactive", "somewhat active", "quite active", "definitely active", for instance. JRank would thus accommodate easily such scenarios.

The algorithm's runtime is quadratic in the size of the training set. This is because the computation of the prediction function involves an iteration through the entire set, in order to compute the similarity between the current input-task pair and the rest of the pairs in the training set. These computation can be cached (since they do not need to be recomputed at each iteration), but this quickly becomes intractable as the number of input-task pairs grows above 10000 (our dataset is much larger than that). Needless to say, computing the Pearson's correlation coefficient is computationally very expensive, as it adds a factor of M (the number of input vectors in the dataset, approximately 16000 in our case) each time one computes the similarity between tasks.

As is the case with MT-NNet, JRank can be adapted straightforwardly to our problem setting. Since our $\mathbf{A}_{(\mathbf{x},\mathbf{t})}$ are essentially binary values, the runtime of the algorithm will be reduced.

3.5 Multi-Task Support Vector Machines

3

The general idea behind JRank–to find a hyperplane that separates two classes–can be taken further by stipulating that this hyperplane should be as far as possible from the two different classes, in the feature space defined by Ψ. Assume, as shown in Figure 3.3, that we have managed to find a candidate hyperplane that separates the two classes such that the distance from it to the closest input-task pairs from

either class is the same. Then it can be shown [70] that for the decision function specified by this hyperplane there will be an upper bound on the generalizatio performance, which depends on the margin obtained with it.

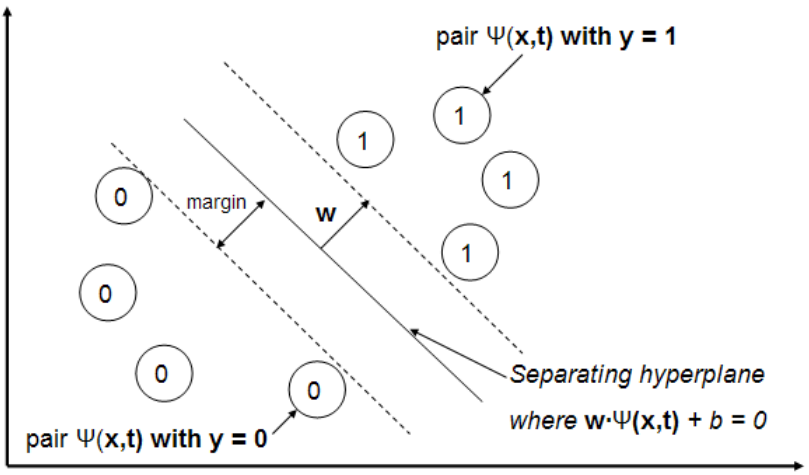


Figure 3.3: Intuition behind the Multi-task Support Vector Machine approach

JRank, as desribed above, will not generally find this hyperplane, because the objective function is non-convex. There is however an algorithm that can efficiently find this hyperplane and its generic name is the Support Vector Machines (SVMs; the "support vectors" are those transformed data that are closest to the optimal hyperplane). Because we are using this algorithm to find separating hyperplanes in joint feature spaces of inputs and tasks, we call our version the "Multi-Task Support Vector Machine".

Let us formalize the intuitions. As above, we are interested in a linear separator, therefore we are looking for a weight vector $\mathbf{w}$ that minimizes some loss, that in turn uses the following decision function:

$$f(\mathbf{x}_i^k, \mathbf{t}^k; \alpha) = b + \mathbf{w}^t \cdot (\mathbf{x}_i^k, \mathbf{t}^k) = b + \sum_{(\mathbf{x}_j^m, y_j^m, \mathbf{t}^m)} \alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)} \cdot y_j^m \cdot (\mathbf{x}_i^k, \mathbf{t}^k)^t \cdot (\mathbf{x}_j^m, \mathbf{t}^m) \quad (3.8)$$

Using kernels, we can find this hyperplane (the plane perpendicular to weight

vector $\mathbf{w}$) in a non-linear transformation of the feature space. To do so, we re-write equation 3.4 into the following decision function:

$$f(\mathbf{x}_i^k, \mathbf{t}^k; \alpha) = b + \mathbf{w}^t \cdot \Psi(\mathbf{x}_i^k, \mathbf{t}^k) = b + \sum_{(\mathbf{x}_j^m, y_j^m, \mathbf{t}^m)} \alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)} \cdot y_j^m \cdot K((\mathbf{x}_i^k, \mathbf{t}^k), (\mathbf{x}_j^m, \mathbf{t}^m)) + b \tag{3.9}$$

These are the so-called dual representation of SVMs. We can see that, as in the JRank case, in these decision functions, data appear only in the form of kernels evaluated at *pairs* of data-points or dot-products of data pairs (a dot-product is a linear kernel, as well).

The basic idea of the Support Vector Machine algorithm is that the procedure for choosing the parameter matrix α is one that gives rise to a hyperplane that has a special property: it maximizes the margin between it and the data from the two classes (so this is the "loss function" that it tries to minimize). The actual equation / objective function that is to be minimized is [65]:

$$\sum_{(\mathbf{x}_j^m, y_j^m, \mathbf{t}^m)} \alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)} - \frac{1}{2} \sum_{(\mathbf{x}_i^k, y_i^k, \mathbf{t}^k), (\mathbf{x}_j^m, y_j^m, \mathbf{t}^m)} \alpha_{(\mathbf{x}_i^k, \mathbf{t}^k)} \alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)} y_i^k y_j^m K((\mathbf{x}_i^k, \mathbf{t}^k), (\mathbf{x}_j^m, \mathbf{t}^m)) \tag{3.10}$$

subject to $\alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)} \geq 0$ and $\sum_{(\mathbf{x}_j^m, y_j^m, \mathbf{t}^m)} \alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)} \cdot y_j^m = 0$. This is a quadratic optimization problem: it is convex and it can be solved in polynomial time. Schölkopf and Smola [65] provide several algorithms for doing so efficiently.

The separating hyperplane might not be able to perform a perfect separation, i.e. without mislabeling any training examples, even in a non-linear high-dimensional space. Moreover, even if the separation is perfect, it could suffer from the problem of overfitting—the hyperplane could be very close to the data and the solution found by it would therefore have a poor bound on the generation error. One could thus introduce some slack variables that allow for separating hyperplanes that allow for mislabeled training examples. These variables can be summarized into a single constraint that introduces a trade-off between the margin of the separation and the number of mislabeled examples. This constraint is a box-constraint

on α: $0 \leq \alpha_{(\mathbf{x}_j^m, \mathbf{t}^m)} \leq C$, with C being the so-called *soft-margin* parameter that controls the trade-off just described.

As we mentioned before, the main difference between JRank and MT-SVM is that the training procedure for MT-SVM produces a maximum-margin separating hyperplane, which is, under the assumption that a maximum-margin is desirable, the best-case scenario of JRank. JRank is however faster and should in theory produce solutions that are close to the MT-SVM solution (especially when used with non-linear kernels).

The algorithm just presented is the standard description of the Support Vector Machines. It uses the custom kernels that unify the inputs and tasks into one joint feature space in which we can compute efficiently similarities between input-task pairs. There are many more details of the implementation that we glossed over, but they are standard issues that arise when using the standard SVM algorithm. We used an off-the-shelf implementation of the algorithm— [42] provides extensive details about it.

We mentioned that this is a quadratic optimization problem (i.e. the criterion to be minimized is a polynomial of degree 2 in the parameters). Depending on the implementation and on the data, the complexity is somewhere between cubic and quadratic in the number of input-task pairs. The algorithm is also linear in the size of the training set at the testing stage: in order to classify an input-task pair into either of the classes, we need to compute the similarity of the pair with all the rest of the pairs from the training set (JRank suffers from the same problem).

The choice of kernels is the same as with JRank and for the same reasons. As described above, MT-SVM will not be able to perform ordinal regression. One could extend the training procedure in order to perform multi-class classification (in a *one vs. all* setting), but this would be only a very rough approximation to the idea on ordinal regression. The ability to do the latter is one of the advantages of JRank.

MT-SVM can be readily applied to the computational chemistry dataset. Essentially, one only needs to write a wrapper function that computes the similarity

measure between input-task pairs and the result will be a decision function that will separate in an optimal way active and inactive molecules in the joint molecule-target feature space.

3.6 Partial Least Squares

Partial Least Squares (PLS) is a very popular algorithm in the computational chemistry research community, partly because it is easy to implement and because it serves as a baseline for comparison. It combines some techniques from Principal Component Analysis with ones from ordinary linear regression. If we assume that we have access to a matrix of inputs $\mathbf{X}$ and (in the most general case) a matrix of corresponding desired outputs $\mathbf{Y}$ (in our case, since there's only one output per input, this will be a vector) then the goal of PLS will be to find a set of latent components that performs a decomposition of both $\mathbf{X}$ and $\mathbf{Y}$ such that these components explain most of the covariance between $\mathbf{X}$ and $\mathbf{Y}$.

We have only used PLS for single-task prediction problems, therefore we do not provide more details of it in this thesis. Wold et al [72] show however its inner workings in more detail.

CHAPTER 4

EXPERIMENTS

In this chapter, we describe the experiments that we performed using the algorithms from Chapter 3

4.1 Experimental Setup

First we provide a technical description of the dataset that we used, as well as most of the details related to our experimental setup and results. Because of the proprietary nature of the dataset, it is not possible for us to give *all* the details needed for reproducing our results.

4.1.1 Datasets and Descriptors

All the ligands (molecular compounds) used in this study were in the possession of AstraZeneca R&D Montreal. They all satisfy the *Lipinski rule of five*[1] [51]. Different subsets of these molecules have been screened against 24 biological targets. The screens have been made at different times, with some small variations in protocol. Our dataset is thus a collection of disparate HTS campaigns brought together for this study. Figure 4.1 shows the number of compounds for which the screening data was available for each target. We detailed the active and inactive compounds.

The compounds descriptors were computed with MOE (version 2004.03) [17] for each of the molecular compounds[2]. The set of 469 descriptors range from atom frequencies [11, 56] and topological indices [3, 35, 44, 58] to 3D surface area descriptors. We also computed MACCS [25], Randic [59] and EState [36] descriptors that

[1] A set of rules of thumb that indicate whether a molecule is likely to be active or not

[2] A forthcoming technical report will give more details about the procedure. This report will be written by the computational chemistry specialists that constructed the dataset and will be available for download from the webpage of our laboratory, `www.iro.umontreal.ca/~lisa`

are available in the MOE package. The numerical values were normalized to zero mean and unit variance.

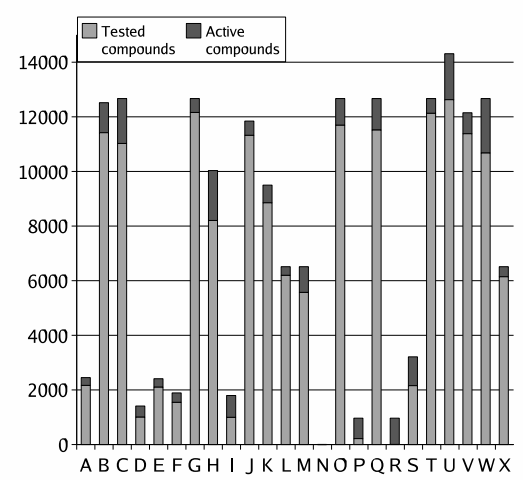


Figure 4.1: Number of compounds available for each target, classified as actives or inactives

We have also been provided with target-specific features/descriptors that we used in the multi-task learning process. The distinguishing features of the biological targets from the same family are the shape and the properties of the so-called binding pocket. Several fingerprints of such pockets have been published in the computational chemistry literature. The ones we used in this work were based on our observations and some assumptions. The first assumption is that all the targets in our study share a similar binding position. The second assumption is that the amino acids of the targets in binding sites have three native interactions between their side chains: ionic, polar, and hydrophobic. When a ligand interacts in the binding site, it will break some of the native interactions and build up new, ligand involved (mediate) interactions. Based on the positions (at the binding site), the type of the interactions, and the variations amounts of the targets in this study, 14 bins were identified and used. Each bin represents a type of the interaction at the given position of the binding pocket. Adding, reducing, or changing the targets will alternate the binding pocket fingerprints.

In order to accommodate for the idea that the algorithms should be able to generalize well to an unknown target, we did not intend to further detail the differences between the targets, which would have been possible with other protein fingerprints [22]. As a final step, we selected the most varying receptor descriptors to match the small number of targets we studied.

The learning methods that we employed take advantage of the prior knowledge about the receptors. The idea is to choose a representation of each receptor and to train a model to predict a single scalar (e.g. probability of percentage inhibition) given both the representation of the compound and the representation of the target receptor protein. Because the representation of receptors is generic (and can accommodate receptors other than those for which assay results are available), this approach can in principle generalize to new receptors.

4.1.2 Task Selection

One of the first problems that we encountered when dealing with the dataset is the sheer size of it. There are more than 186000 test results and two of the algorithms that we used (the ones that performed better) take more than quadratic time in the number of the examples, so it made sense to perform some sort of subsampling of the data.

One way of doing that is by selecting the targets for multi-task learning. If we had some simple way of computing the degree of "relatedness" between two targets and selected a subset of $k < 24$ target that are most related to each other, then the data obtained in this way should capture a lot of the "shared representation" between the targets, which we are trying to uncover.

The measure chosen here is the pairwise linear correlation of activity between each target, for shared chemical compounds in the dataset. The linear correlation will get higher when two targets have the same active and inactive compounds. Section 4.2.1 presents the targets that were selected with this procedure.

4.1.3 Undersampling Scheme

In order to test the efficiency of a multi-target scenario, we need a framework that would provide an estimate of target "relatedness" and an estimate of how much multi-target learning help as compared to single-target learning. Such a framework can then be used to decide whether multi-target learning makes sense in the first place before proceeding to the actual HTS campaign for a new target.

We devised the framework as follows. Assume that we have a set of targets from the same family, with enough screening data for each target. For each of them, we construct two datasets:

1. A training set that contains the screening data for all the targets except the current one *plus* a fixed small percentage of the screening data for the current target.

2. A testing set that contains the rest of the screening data for the current target.

By training an algorithm on the first dataset and testing on the second one and then comparing the performance of this algorithm with the performance of some other algorithm that does standard, single-target, QSAR modeling (with the training set containing just the fixed small percentage of the screening data for the current target), we can see whether adding the screening data for the rest of the targets improves the results.

The reasoning behind choosing a small percentage is simple—we want an algorithm to generalize well given a *new* target, for which we have insufficient screening data, and that is a quite realistic scenario. A standard single-target QSAR model that is trained on a small dataset will most likely have a poor performance; an algorithm that does multi-target learning well (using the above-mentioned training set) should perform no worse or better than such a single-target model.

We call the procedure of making the above datasets "undersampling". Our intention is to try to see the effects of undersampling on both multi-target and

single-target data at several fixed percentages (which we sometimes call "undersampling fractions") of the screening data for the targets in our dataset. Another intention of ours is to see what happens when the undersampling fraction is actually equal to zero, i.e. there is no training data related to the new target. If by training on a set of targets and testing on a new target we obtain better-than-random results, then we can conclude that the learning algorithm can uncover a shared representation of these targets. This would be a very positive result.

One of the assumptions behind our experiments is that the targets are related in some way that is encoded in our dataset. Our goal is to obtain multi-target learning procedures which will be at least as good as single-target learning, and that will outperform single-target learning for small undersampling fractions. We want to test the hypothesis that such a procedure can be successful in the context of multi-target HTS data.

Algorithm 2 describes a simple method for testing the predictive power of a certain multi-task learning algorithms on any of the data from these targets. By artificially depleting the dataset used for training and validation of examples from a certain task and by varying the level of depletion, we can obtain a relatively complete picture of the performance of such an algorithm given different real-life scenarios. Such a scheme also allows for direct comparisons between multi-task/target learning and single-task/target learning.

4.1.4 Hyper-Parameter Selection

In order to assess the generalization performance of the algorithms presented, we need to select a combination of hyper-parameters that gives an optimal performance on a validation set (which is independent from the training set). Given this combination of hyper-parameters, we can get an unbiased estimate of the generalization ability of the algorithms by testing the models on a testing set, that is independent from both the training and validation sets.

In the case of the neural network, such model selection procedures have been performed for parameters such as the number of hidden units, the weight decay,

Algorithm 2 A sample scheme of hypothesis testing

```
{T is the total number of tasks}
{R is a real number between 0 and 1 (the "undersampling fraction")}
{K is the number of iterations}
for t = 1 to T do
  for R = 0 to 1 (by increments ΔR) do
    for k = 1 to K do
      {Randomly subdivide the dataset into two parts (the parts in bold only
      apply to the multi-target case):}
      DTrain = {Fraction R of data from task t } + { Data from all tasks
      except t}
      DTest = {The rest of data from task t}
      {Perform training and validation (model selection) on DTest}
      LModel = Train({Single,Multi}TaskAlgo, DTrain)
      {Compute the error of the selected model on the test set}
      Error(T,R,k) = Test(LearnedModel,DTest)
    end for
  end for
end for
```

the learning rate, etc. Early stopping on the number of epochs (by computing the validation error at each step and stopping when it starts to increase) has also been performed.

In the case of JRank, the width of the Gaussian kernels was computed by the above validation procedure, whereas early stopping was used for finding an optimal N_{it}. The optimal width of the Gaussian kernels and the soft-margin parameter C for MT-SVM were computed using the same validation procedure (but no early stopping in this case).

4.1.5 Performance Measures

We use the LIFT to assess the performance of the models. The LIFT measures how much better than random we can order the compounds from active to inactive. This measure is quite useful for those datasets that are very unbalanced and where a baseline algorithm (that simply predicts the larger class all the time) would have a very good performance accuracy-wise. We compute the LIFT by testing a model

on an independent test set and ordering (in decreasing order) the molecules by the scores that we obtain for each of them. We select a subset of this ordered list (from the highest ranking molecule downwards) and compute ratio of actives to the total number of compounds in this subset. The higher this ratio, the better is our algorithm at predicting which compounds are active.

Let a/n be the average fraction of actives in our database, with a the total number of actives, n the total number of compounds (this fraction is a value that is close to 0.1 in our dataset). In the selected subset, the one that we tested the model on, a_s is the number of actives and n_s the number of compounds in that subset (and it is hoped that this fraction will be higher than a/n). Then we compute the LIFT as

$$\text{LIFT} = \frac{a_s/n_s}{a/n} \tag{4.1}$$

In effect, the LIFT tells us how much better than chance our algorithm performs. The LIFT that we compute is a single point in an enrichment curve that corresponds roughly to an ROC curve [37]. The enrichment curve tracks the LIFT values across different sizes of the subsets and provides a comprehensive picture of the generalization capabilities of a learning algorithm; it can also be transformed straightforwardly into an ROC curve [50]. Here, the subset is 30% of the database, which is one of the standard values in the computational chemistry literature, and we multiply the LIFT values by 100 to improve readability.

4.1.6 Target descriptors influence

We were interested in finding out how much the target descriptors influence the decision making process, as a function of the undersampling fraction. We had expected that this influence would decrease as we add more and more data from the given target and that it would be maximal when the undersampling fraction is small. This estimate can be computed by simply observing that, if the task kernel and the data kernel are both Gaussians, then the resulting task-datapoint

kernel will be Gaussian as well (as per equation 3.7). The exponent will be a sum that has a term for the task and a term for the input. The relative fraction of the task-related term (when applying the kernel, for instance, at the test stage) will give us an idea of the level of influence of the task features.

More formally, assume that the kernel for the input features is:

$$K(\mathbf{x}_i^k, \mathbf{x}_j^m) = \exp\left(-\frac{||\mathbf{x}_i^k - \mathbf{x}_j^m||}{2\sigma_x^2}\right) \tag{4.2}$$

and the one for the task descriptors is:

$$K(\mathbf{t}^k, \mathbf{t}^m) = \exp\left(-\frac{||\mathbf{t}^k - \mathbf{t}^m||}{2\sigma_t^2}\right) \tag{4.3}$$

then the similarity between the (input,task) pairs $(\mathbf{x}_i^k, \mathbf{t}^k)$ and $(\mathbf{x}_j^m, \mathbf{t}^m)$ is

$$K((\mathbf{x}_i^k, \mathbf{t}^k), (\mathbf{x}_j^m, \mathbf{t}_m)) = K(\mathbf{x}_i^k, \mathbf{x}_j^m) \cdot K(\mathbf{t}^k, \mathbf{t}^m) = \exp\left(-\left(\frac{||\mathbf{x}_i^k - \mathbf{x}_j^m||}{2\sigma_x^2} + \frac{||\mathbf{t}^k - \mathbf{t}^m||}{2\sigma_t^2}\right)\right) \tag{4.4}$$

It is clear now that by analyzing the proportion of $\frac{||\mathbf{t}^k - \mathbf{t}^m||}{2\sigma_t^2}$ in the above exponential, we can figure out the degree of influence of the task descriptors. This degree of influence is helpful during the testing process: for each undersampling fraction and for each pair of tested (input,task) we can evaluate the above kernel formula and compute

$$\text{influence}_{tfraction} = \sum_{i,j,k,m}^{N_{tm},N_{trm},N_{tt},N_{trt}} \frac{\frac{||\mathbf{t}^k - \mathbf{t}^m||}{\sigma_t^2}}{\frac{||\mathbf{x}_i^k - \mathbf{x}_j^m||}{\sigma_x^2} + \frac{||\mathbf{t}^k - \mathbf{t}^m||}{\sigma_t^2}} \tag{4.5}$$

where σ_x^2 is the learned width of the input features Gaussian kernel, σ_t^2 is the task descriptors equivalent, $N_{trm} \times N_{trt}$ is the number of support vectors resulting from the training process (the number of (input,task) pairs that are on the margin) and $N_{tm} \times N_{tt}$ is the number of tested (input,task) pairs.

4.2 Experimental Results

We can now present the results obtained with the techniques described in Chapter 3 and the setup described in Section 4.1

4.2.1 Task Selection

Figure 4.2 shows the number of compounds for which we have pairwise screening information. As can be seen, for many pairs, the shared number of compounds is quite small. It should be noted that the choices of compounds for the screening introduce some unnecessary biases in a multi-target scheme, but it is a necessary procedure, at least for now, as it makes our computations feasible.

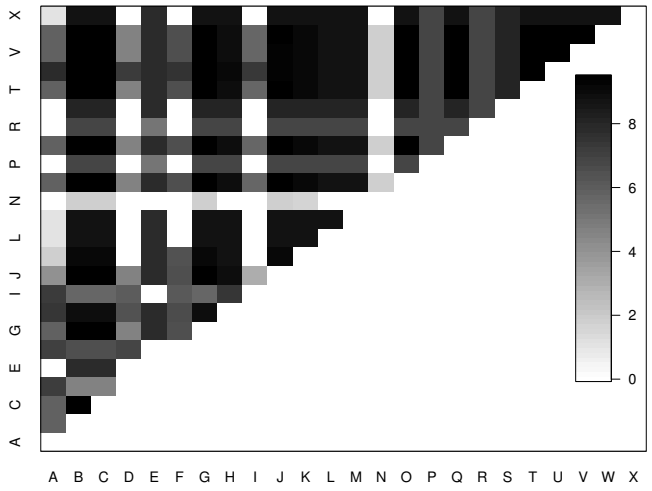


Figure 4.2: Number of compounds shared by each pair, in logarithmic scale

Figure 4.3 shows the actual pairwise linear correlation of biological activity. From this, we picked 7 targets, for which cross-correlation where the strongest. These seven targets, G1A, G1D, G1F, G1H, G1I, G1S and G1U (they correspond to 45000 target-compound pairs in our dataset) will constitute the main focus of

interest for the undersampling experiments.

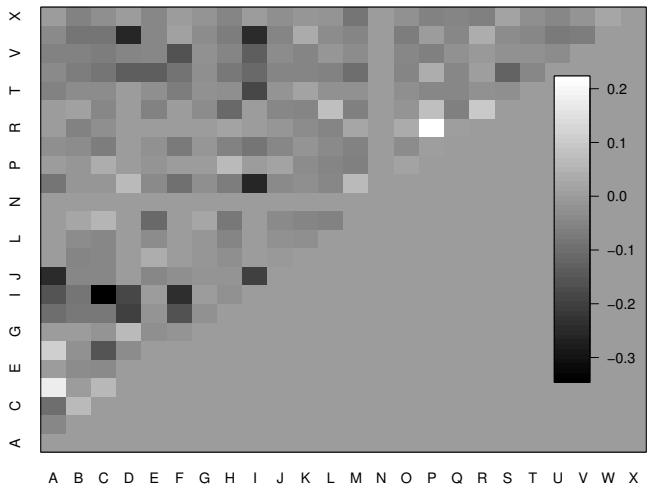


Figure 4.3: Pairwise Correlation of Biological Activity

4.2.2 Multi-Task Neural Network

One of the first experiments that we have performed is varying the number of targets in the dataset and measuring the generalization performance of the algorithm. The targets were selected using the method described in 4.1.2. So, for instance, column 2 of table 4.1 contains the generalization performance of an MT-NNet that was trained in turn on 11 targets and tested on the remaining one, with the training set containing 90% of this target data (in addition to the data from the rest of the targets) and the test set containing the rest of 10%. We have reproduced the results for the case of learning being done using the target descriptors and without them. Interestingly, the performance seems to increase as we add more targets, but decreases as we continue doing so. This can be explained by the fact that 3 targets are simply not sufficient for finding and exploiting the shared representation between the targets. Whereas 7 seems to be the optimal choice, 12

Table 4.1: Comparing MT-NNet's performance with and without target descriptors

LIFT	**24 targets**	**12 targets**	**7 targets**	**3 targets**
Without TD	179	171	195	188
With TD	189	175	195	193

Lift over 30% of data

and 24 bring up poorer performance because, in our opinion, they (the 12 minus 7 and 24 minus 7 targets) are too unrelated and, effectively, add "noise" to the learning procedure.

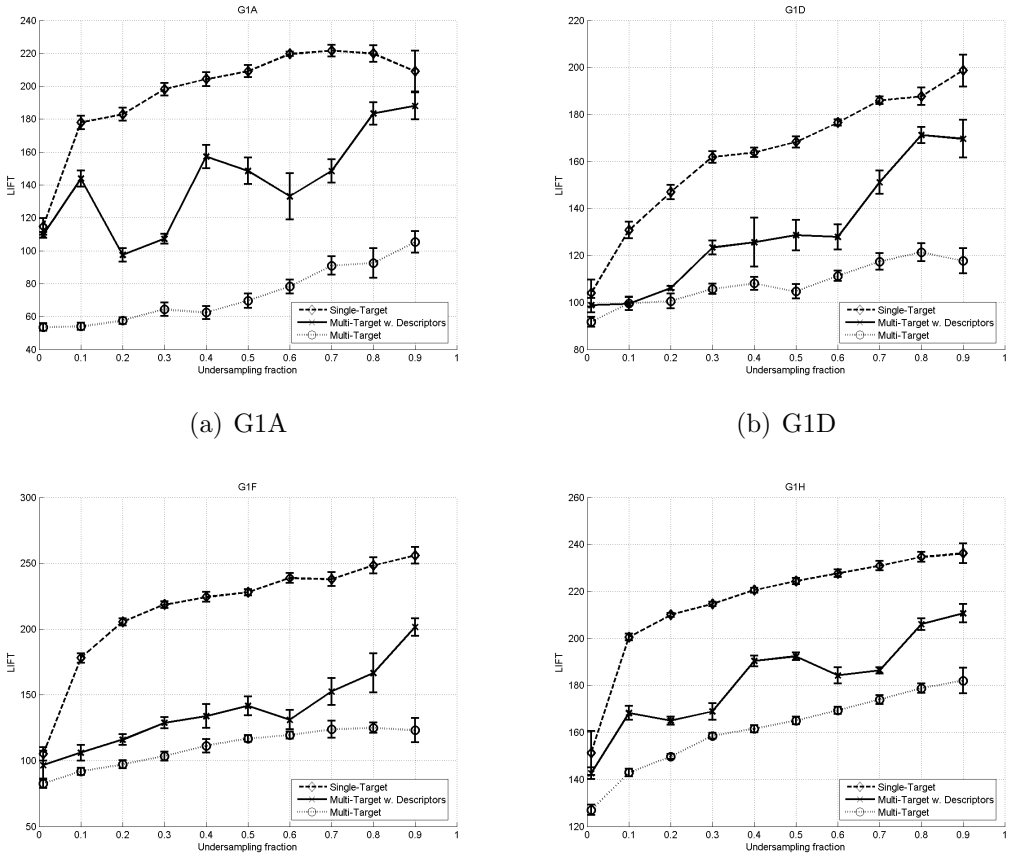
Another interesting point is the fact that the performance when using target descriptors is consistently better than when not using them. This is quite encouraging, as it supports our further experiments.

Figures 4.4 to 4.5 show the details of undersampling the 7 most correlated targets with the neural network. We test our algorithm with and without target descriptors and compare with single target learning when done with the same neural network. We see that the LIFT rises quickly when doing single target learning, and that multi-target learning without target descriptors lags far behind. Depending on the target, multi-target learning with target descriptors falls in between. For G1I, we even see a slight range of undersampling where multi-target learning beats single target learning.

One of our hypotheses–that target descriptors seem to help with learning–seems to hold true: Overall, however, the results are disappointing, since there is hardly any range of undersampling fractions for which MT-NNet beats single-task neural networks.

4.2.3 JRank

We performed the same type of experiments with JRank. We found that using a combination of identity (for regularization) and Gaussian kernels produced the best results. The correlation kernel did not seem to capture too well the similarities between pairs of targets or pairs of compounds (a possible reason is that the



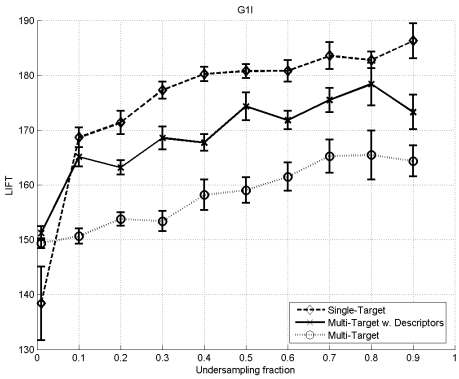
(a) G1A

(b) G1D

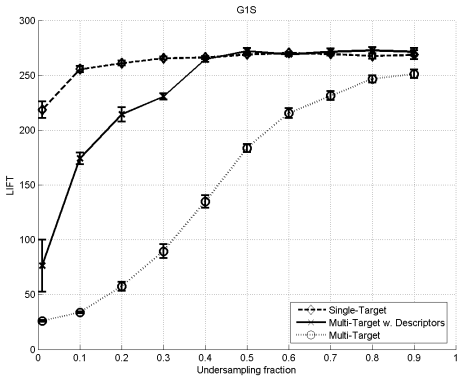
(c) G1F

(d) G1H

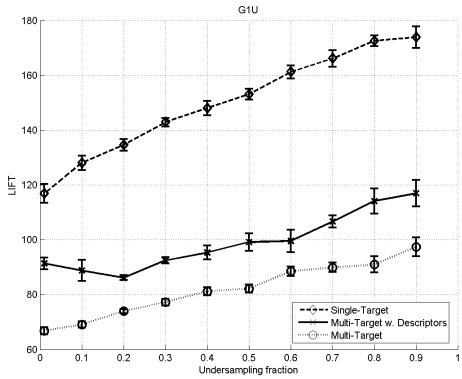
Figure 4.4: Effects of undersampling on Neural Net's performance. Measured on the G1A, G1D, G1F and G1H targets in 3 scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning



(a) G1I



(b) G1S



(c) G1U

Figure 4.5: Effects of undersampling on Neural Net's performance. Measured on the G1I, G1S, G1H targets in 3 scenarios: multi-target learning withtarget descriptors, multi-target learning without target descriptors, and single-target learning

simple linear correlation coefficients between either targets or compounds are not sufficient to capture any similarity measure between them) and its computational price hinders extensive experimentation.

Figures 4.6 through 4.7 contain the results obtained with this algorithm. This time it seems that in most of the cases multi-target learning with target descriptors is at least as good as single-target learning or multi-target learning without descriptors and that, in one case (Figure 4.6), it seems to perform better than either of them.

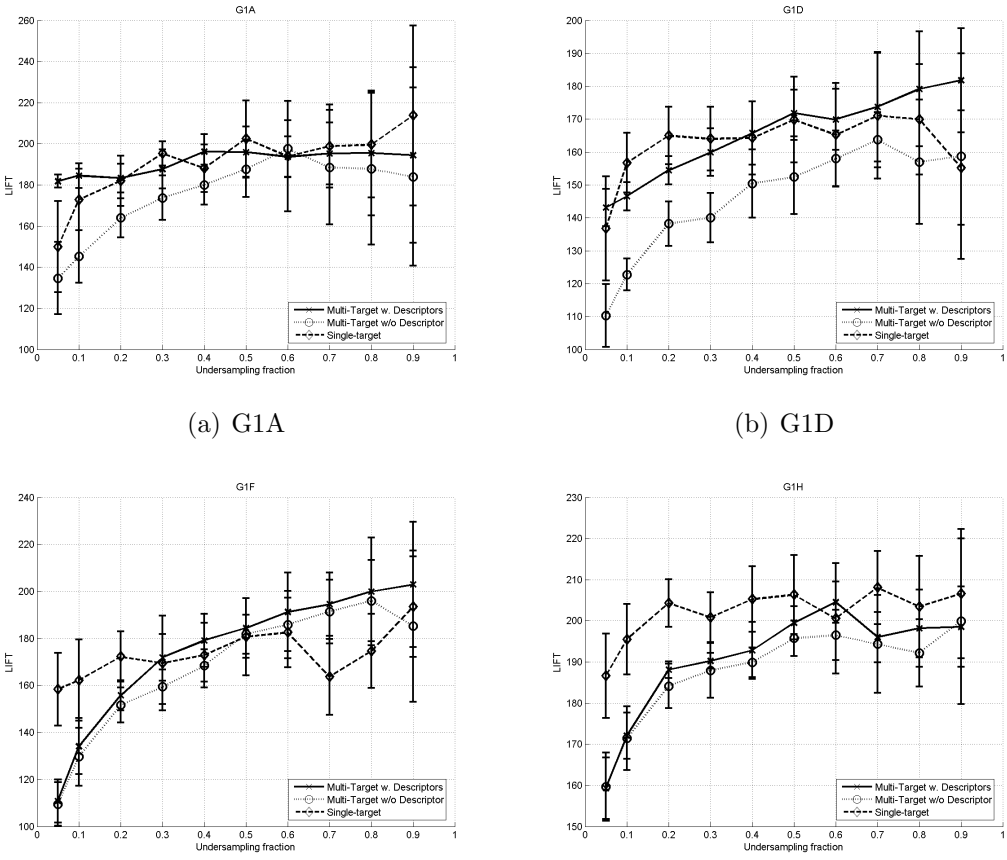
By comparing figure 4.4 and 4.5 with 4.6 and 4.7 we clearly see that JRank scores much higher than the neural network on the smaller fraction of undersampling. Clearly, JRank is to be preferred to MT-NNet in a multi-target setting. We also notice that JRank performs quite well even in a simple single-target scenario and therefore could be used in a stand-alone fashion.

4.2.4 Multi-Task Support Vector Machines

While the results obtained with JRank were indeed encouraging, Figures 4.8 through 4.9 show that there is potential for more. These figures contains the results obtained with undersampling the 7 targets with MT-SVM. In Section 4.2.7 we will see that MT-SVM consistently beats JRank results across all targets. The other encouraging fact is that MT-SVM is as good as ST-SVM across all targets; given that ST-SVM is considered to be the state-of-the-art in computational chemistry/drug-discovery research, this is quite a positive result, but a disappointment as well, since MT-SVM does not outperform ST-SVM.

4.2.5 Target Descriptors' Influence

As per Section 4.1.6, we have also computed an estimate of the "degree of influence" of target descriptors. Figures 4.10 and 4.11 show such plots for the 7 targets. Several of the plots confirm our expectations, but they make us doubt the quality of the descriptors that we received (and of the resulting learned predictors



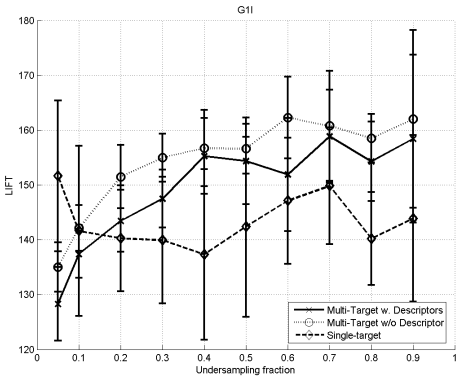
(a) G1A

(b) G1D

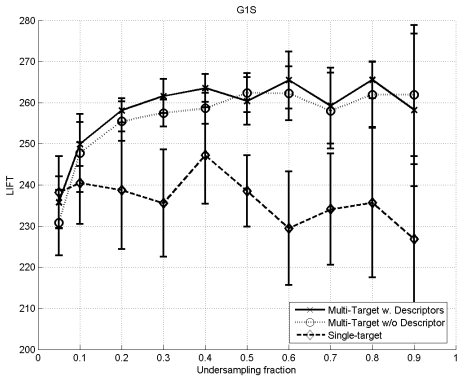
(c) G1F

(d) G1H

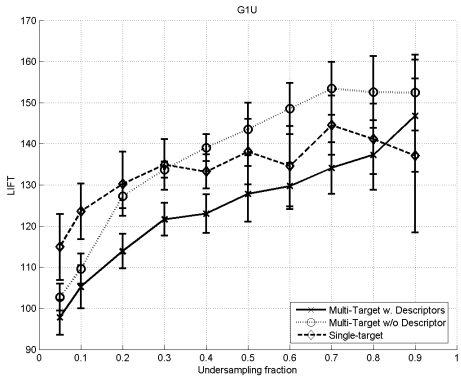
Figure 4.6: Effects of undersampling on JRank's performance. Measured on the G1A, G1D, G1F and G1H targets in 3 scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning



(a) G1I

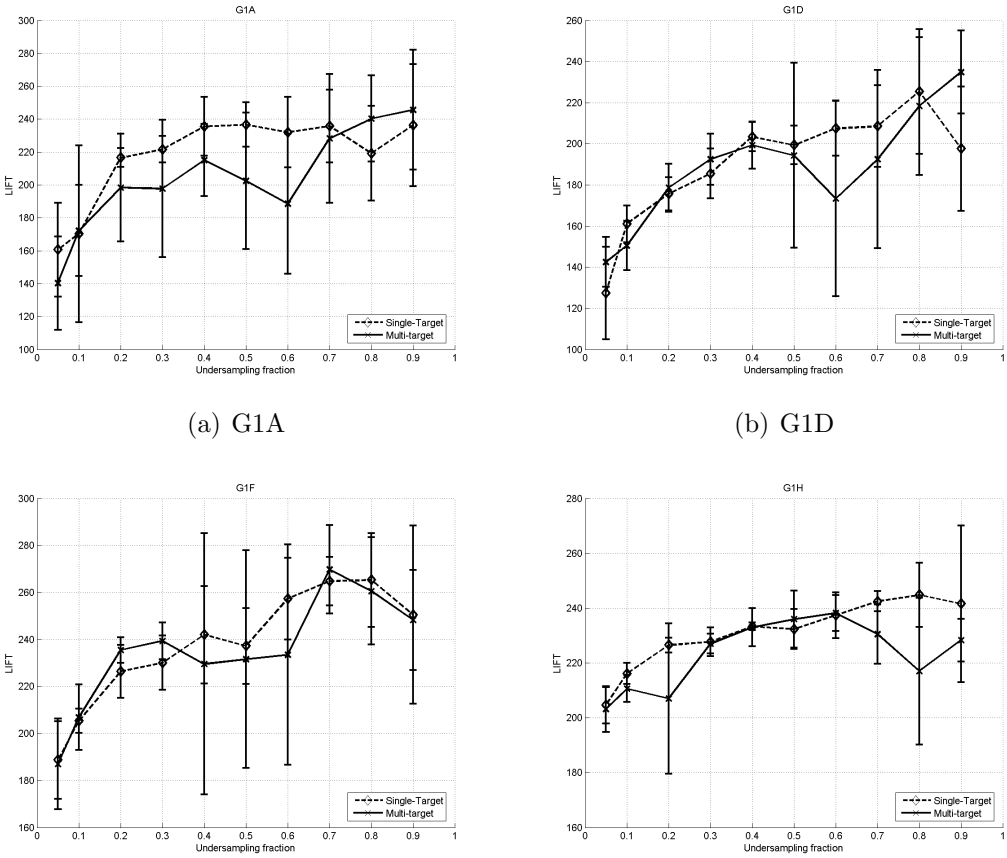


(b) G1S



(c) G1U

Figure 4.7: Effects of undersampling on JRank's performance. Measured on the G1I, G1S, G1H targets in 3 scenarios: multi-target learning withtarget descriptors, multi-target learning without target descriptors, and single-target learning



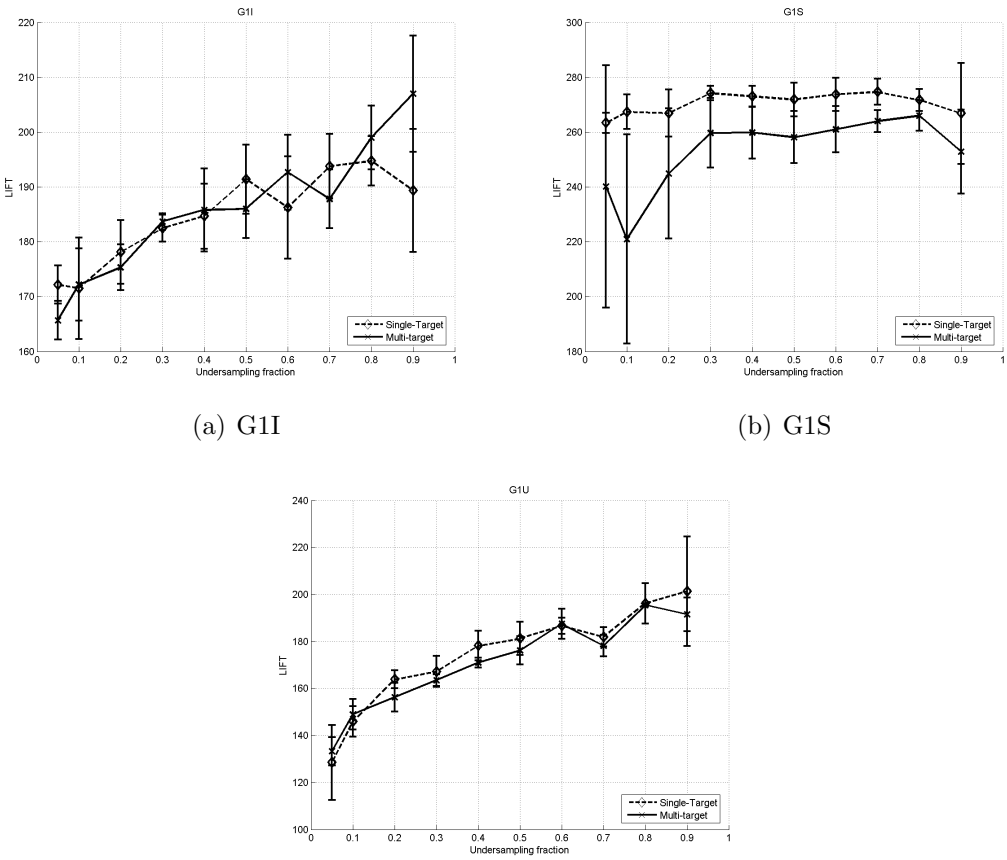
(a) G1A

(b) G1D

(c) G1F

(d) G1H

Figure 4.8: Effects of undersampling on SVM's performance. Measured on the G1A, G1D, G1F and G1H targets in 2 scenarios: multi-target learning with target descriptors and single-target learning



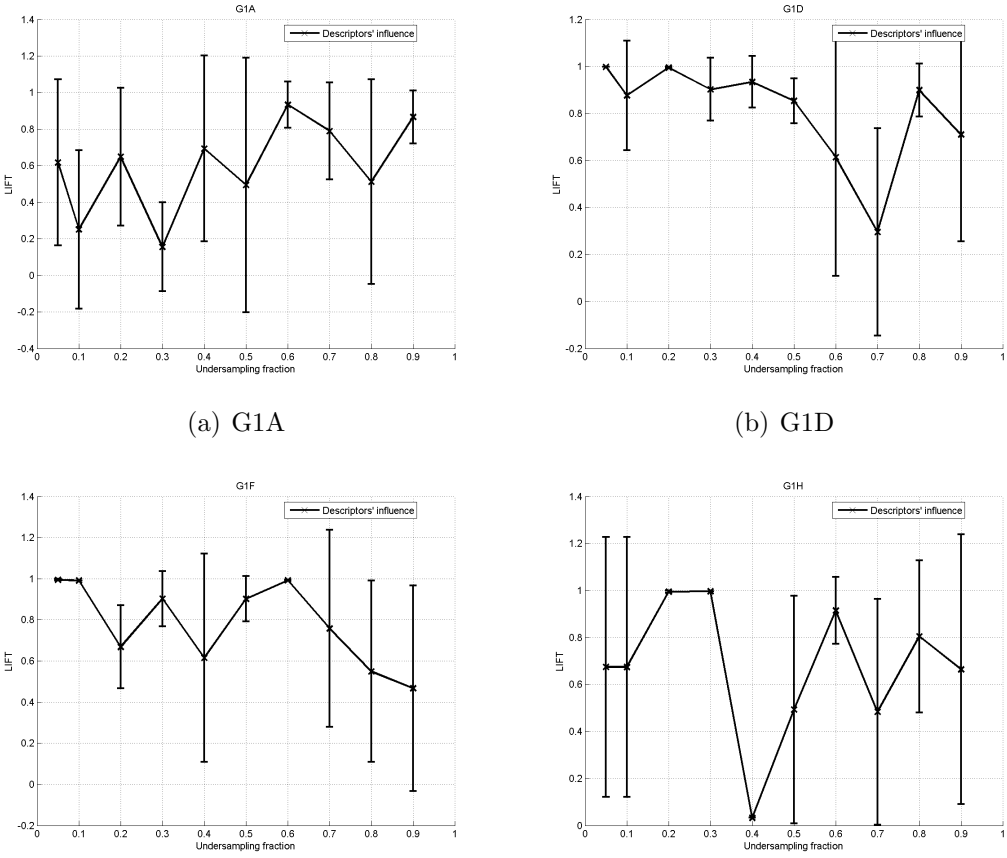
(a) G1I

(b) G1S

(c) G1U

Figure 4.9: Effects of undersampling on SVM's performance. Measured on the G1I, G1S, G1H targets in in 2 scenarios: multi-target learning with target descriptors and single-target learning

that depend on them).



(a) G1A

(b) G1D

(c) G1F

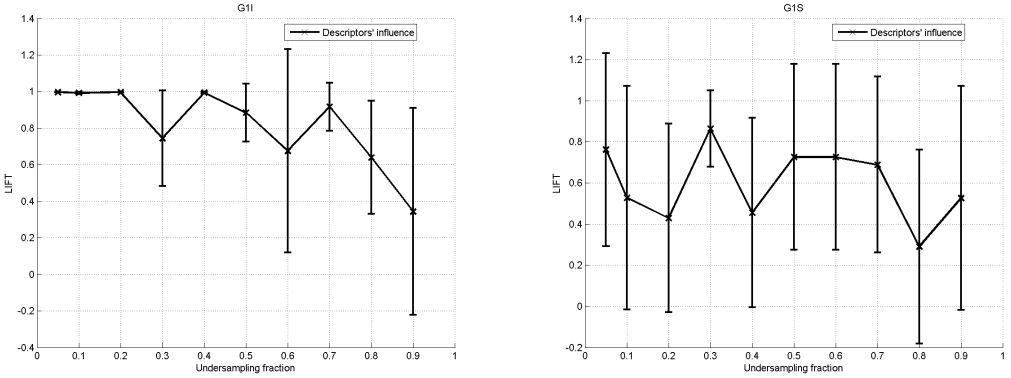
(d) G1H

Figure 4.10: The influence of the target descriptors when an MT-SVM model is trained on a set of data for the respective targets, G1A, G1D, G1F and G1H, which varies in size

However, these plots should be taken with a grain of salt, as the influence measure that we just described is quite an heuristic. A more theoretically sound and more reliable measure is certainly desirable.

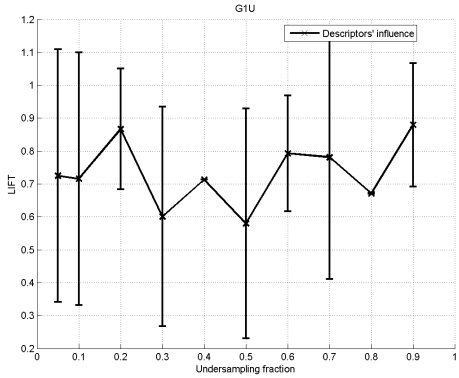
4.2.6 Zero-data experiments

Our final experiments concern the more fundamental question of whether inductive transfer is at all possible. One quite simple way of testing that within our



(a) G1I

(b) G1S



(c) G1U

Figure 4.11: The influence of the target descriptors when an MT-SVM model is trained on a set of data for the respective targets, G1I, G1S and G1U, which varies in size

framework is by setting the undersampling fraction to zero–i.e. training on $N-1$ targets and testing on the Nth target.

Table 4.2 shows a summary of the results. For 3 out of 7 targets, the Multi-Task Support Vector Machine generalized quite well, with LIFTs in the range of 130–161 (where 100 is the LIFT of a random decision algorithm). JRank and MT-NNet did not seem to be able to do the same, with LIFT values hovering around the 100 value.

The results obtained with MT-SVM are one of the key findings of this thesis: they show pretty clearly that in the case of MT-SVM and of this dataset, it is *possible* to transfer some of the "knowledge" acquired from learning a set of tasks to a completely new one with zero training examples.

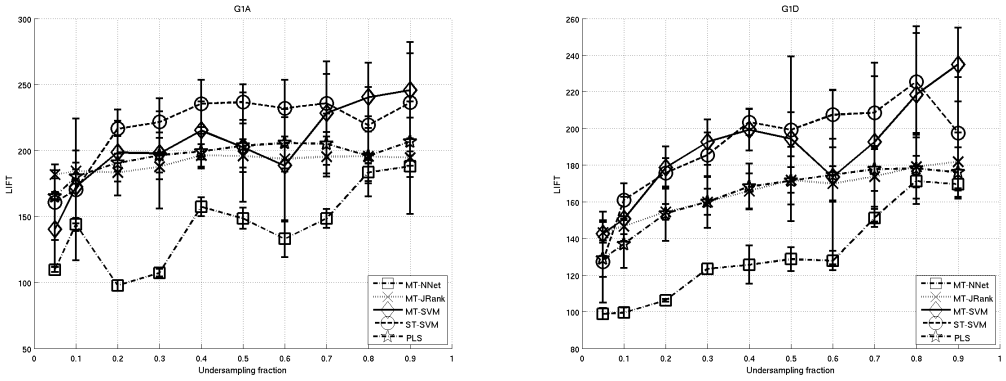
Target	**MT-SVM**	**JRank**	**MT-NNet**
A	105	102	102
D	108	99	95
F	**150**	108	101
H	**161**	110	105
I	**130**	104	103
S	106	105	102
U	105	105	98

Table 4.2: Lifts obtained by testing on a completely new target with no training data

4.2.7 Comparison of all algorithms

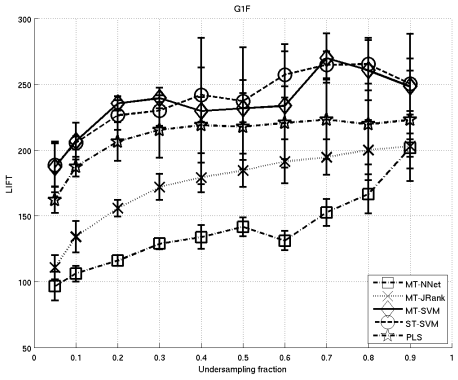
Finally, we compare all the techniques that we have employed, with each other and with a popular (in the computational chemistry industry and research) baseline algorithm called Partial Least Squares. Figures 4.12 and 4.13 contains these comparisons for each undersampling fraction. Table 4.3 contains these results for an undersampling fraction of 0.9, which corresponds roughly to a 10-fold cross validation. Table 4.4 contains the results for an undersampling fraction of 0.1.

In the first case, MT-SVM and ST-SVM perform best, with PLS coming as a distant third, while JRank and NNet are at the bottom of the performance

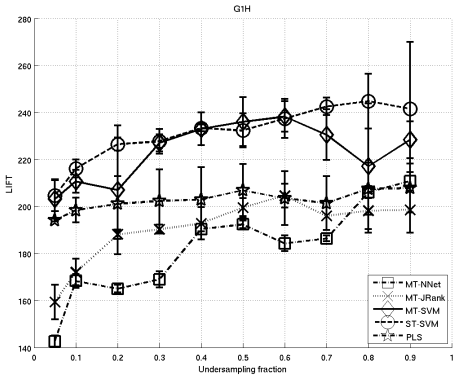


(a) G1A

(b) G1D

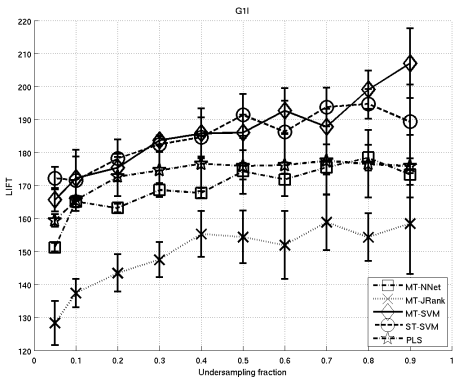


(c) G1F

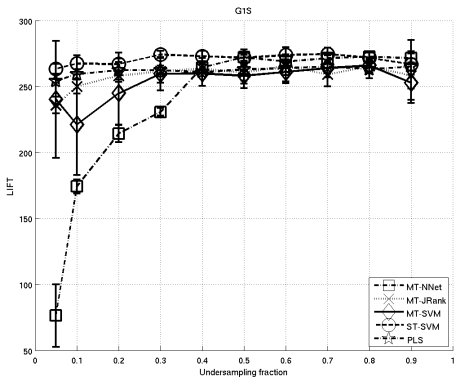


(d) G1H

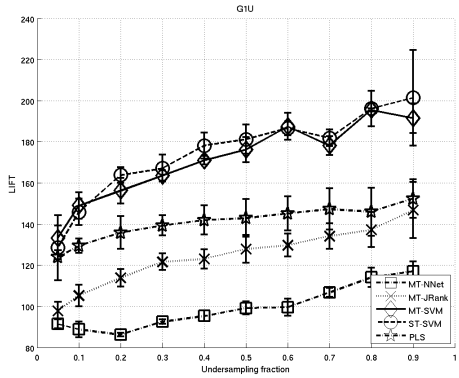
Figure 4.12: Comparison of MT-NNet, MT-JRank, MT-SVM and ST-SVM



(a) G1I



(b) G1S



(c) G1U

Figure 4.13: Comparison of MT-NNet, MT-JRank, MT-SVM and ST-SVM

Target	PLS	ST-SVM	MT-SVM	JRank	MT-NNet
A	195	219	**240**	195	183
D	175	**225**	219	179	171
F	219	**265**	260	199	166
H	207	**244**	217	198	206
I	176	194	**199**	154	178
S	263	271	266	265	**272**
U	145	**196**	195	137	114
Avg	198	**230**	221	189	184

Table 4.3: Comparison of all multi-target methods with ST-SVM and PLS. Lifts computed at tfraction=0.9

Target	PLS	ST-SVM	MT-SVM	JRank	MT-NNet
A	180	170	172	**184**	143
D	136	**161**	150	146	99
F	187	205	**206**	134	106
H	198	**216**	210	172	168
I	165	171	**172**	137	165
S	259	**267**	221	249	174
U	129	145	**149**	105	88
Avg	179	**190**	182	161	134

Table 4.4: Comparison of all multi-target methods with ST-SVM and PLS. Lifts computed at tfraction=0.1

list. In the second case, which is closer to the specifications of our project (that undersampling should be small), the order is the same, except that the difference between the SVM-based algorithms and PLS and JRank is not that big.

CHAPTER 5

DISCUSSION AND FUTURE WORK

5.1 Discussion

Building a virtual screening model for a new target is a difficult task. We developed a special kind of neural network that used a collaborative filtering approach to address the problem. We were disappointed by the poor results. We have nevertheless used the current target descriptors (developed by AstraZeneca), which need a minimal knowledge of the 3D structure of the target. These target descriptors helped to improve the predictive performance and proved that adding new targets helped the learning.

We then implemented and evaluated a kernel-based algorithm, JRank, to address our multi-task problem and to try to cast the problem from a kernel point of view. We presented evidence that JRank is better than our neural network architecture in both single and multi-task settings.

Even if JRank outperforms the Neural Network architecture, its performance still falls short of the performance of the two most common baseline algorithms, which are used extensively in the computational chemistry literature, PLS and ST-SVM. We have thus attempted to apply a modified version of the classical SVM algorithm, using the custom collaborative filtering-inspired kernels, to our problem. The results that we obtained are much more encouraging–MT-SVM outperforms all the algorithms except ST-SVM, with which it is on par.

The main disadvantage of MT-SVM is the time it takes for the model to be trained. Given the enormous size of the dataset that comes out of transforming the data into a form that is suitable for learning with it, training requires significant computational resources (several hours per *task/undersampling fraction/one choice of the hyper-parameters*). ST-SVM obviously does not suffer from this problem, at least not to this extent. Computationally speaking, JRank is faster than both of

the algorithms (due to its online nature), with MT-NNet and PLS being the fastest of all.

The most encouraging results of all is the fact that MT-SVM can generalize in a zero-data scenario, i.e. it generalizes to a new task/target without ever being presented with samples from the dataset of that task/target. This reinforces our belief that having better task descriptors would probably change things for the better and that MT-SVM (and, quite possibly, the rest of the algorithms that rely on the descriptors) would be able to perform better in the undersampling scenario. We have attempted to obtain other task descriptors, but the proprietary nature of the dataset makes this rather difficult. In theory, good task descriptors *must* help; if they allow a learning algorithm to discriminate between tasks and, therefore, compute a reliable estimate of similarity between them, and if we assume that similarity in the descriptor space corresponds to similarity in the predictions then we should be able to "transfer" knowledge from one task to another.

If we view the problem of task descriptors from the opposite angle, then if one measures dissimilarity between tasks using a kernel, intuitively, the "inductive transfer" between them will not happen if they are too far apart in the feature space (as measured by the kernel). Which brings us to a plausible answer to the question of why MT-SVM did not perform better than ST-SVM: either the tasks are too far apart in the feature space or they are too close and it is not possible to do any sort of "clustering" based on the descriptors. If, in addition to that, the multi-task hypothesis–that similarity in feature space corresponds to similarity in the predictions–does not hold true for our dataset (this is quite likely, in our opinion) then all of these could be factors that explain the relatively surprising results that we obtained.

5.2 Future Work

We have obviously not explored all possible solutions to the multi-task problem. In the following, we list some possible refinements to our techniques, which in our

opinion could improve the results that we obtained and offer new insights into the multi-task problem.

5.2.1 MT-SVM considerations

There are many possible avenues for improvement of MT-SVM. Here are some of the ideas that we plan on trying in the future:

- Try out different target descriptors. We have already seen that their mere presence helps with learning, therefore if we were to get higher quality descriptors, MT-SVM should perform better, as we argued in the previous section.

- Use data from all 24 targets for training. This could help, but it could also have negative effects, because of the fact that the targets might not necessarily be related (and the noise introduced in this way could be harmful).

- We have not tried other kernels, except the Gaussian one. It is quite possible that a polynomial kernel or a linear combination of several different kernels could help the process.

- The JRank paper [5] presents several different options for kernels, such as the Pearson correlation coefficient that we did not compute because it was too expensive computationally. We have obtained preliminary results that show that in the case of JRank and our dataset, these coefficients are of no help to the performance of the algorithm, but perhaps they could be helpful if used with MT-SVM.

- Another way of generating target descriptors would be to learn the shared representation of targets with MT-NNet and use the learned weights matrix W (which corresponds to a low-dimensional embedding of target descriptors) as target descriptors. Perhaps these descriptors would allow for better similarity computation in a Gaussian feature space.

5.2.2 A neural network extension

One could try extending the neural network architecture, by predicting not only the activity but also the target and compound features. If constructed properly, such a network could profit from the inductive bias of these features. This bias

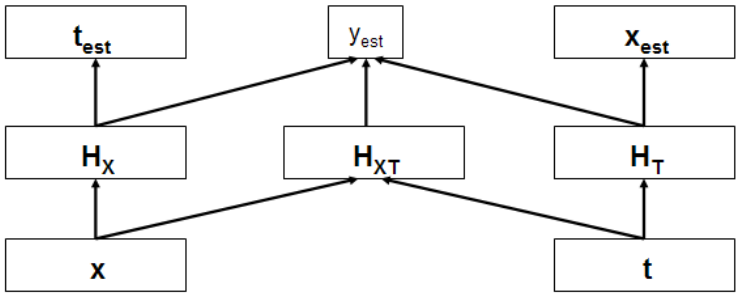


Figure 5.1: The architecture of a possible extension to the neural network

could come in the form of the loss function to be minimized. For an example see Figure 5.1 and this loss function:

$$\mathcal{L}(\mathbf{x}, \mathbf{t}, y) = ||\mathbf{x} - \mathbf{x}_{est}||^2 + ||\mathbf{t} - \mathbf{t}_{est}||^2 + (y - y_{est})^2 \tag{5.1}$$

5.2.3 Other avenues

One could also try to apply the ideas from [28] to this problem: they present an objective function for SVMs which could profit from task/target descriptors. Bakker and Heskes's Bayesian way of multi-task learning [2] can accommodate for such descriptors, too, therefore it is an option to consider for further research. One should consider coming up with a generative model, instead of a purely discriminative one. Intuitively, it seems that a generative model could cope better with the concept of generalizing to a completely new task for which there are no training examples at all. Finally, it would be very useful to find a better, more reliable and theoretically sound measure of the influence of the target descriptors. Since it is

a relatively easy task, a pharmaceutical company can and will invest significant effort into finding better descriptors if this measure can be then used to reliably generalize to a new target.

CHAPTER 6

CONCLUSION

We have evaluated several machine learning algorithms that we used to solve a computational chemistry problem. We were interested in studying the behavior of these algorithms when presented with a completely new and unseen task. Our goal was to design an algorithm that would be able to transfer the knowledge acquired from learning several (possibly related) tasks to a new task. To this end, we designed a Multi-Task Neural Network, tested a kernel-based method called JRank and applied the ideas from JRank to the standard Support Vector Machine algorithm (we call this extension the Multi-Task Support Vector Machine).

The Multi-Task Support Vector Machine algorithm matches the performance of the state-of-the-art algorithm for the given problem, at all undersampling fractions. It also manages to achieve inductive transfer when tested on several unseen tasks/targets, which is an encouraging result. It makes us believe that such inductive transfer, across different biological targets, is possible in general (and not only with these targets). We have suggested possible avenues for improvement and traced parallels to other published algorithms that could make intelligent use of our dataset.

We are quite confident that this is the first time that multi-task learning has been applied to a drug discovery problem and we believe that an improved version of the MT-SVM could in fact influence future practice in this domain. The MT-SVM algorithm is an easy to apply technique, in contrast to the more complex Bayesian methods that could in principle be used to solve the same problem. MT-SVM can also straightforwardly incorporate task-specific descriptors into the learning procedure (indeed, they form the crux of it).

It seems that the task descriptors are crucial in improving the generalization abilities of the multi-task techniques that we considered. We have argued that better descriptors could have led to better results, both in the zero-data case and

in the undersampling case. Even so, the results that we obtained are encouraging and promising. They validate our general approach of using multi-task learning for combining the data from multiple biological targets and they certainly call for refinements of the techniques considered and for future work in this domain.

BIBLIOGRAPHY

[1] T. Anoyama and H. Ichikawa. Neural networks as nonlinear structure-activity relationship analysers: useful functions of the partial derivative methods in multilayer neural networks. *Journal of Chemical Information and Computer Sciences*, 32:592–500, 1992.

[2] Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 4:83–99, 2003.

[3] A. Balaban. Highly discriminating distance-based topological index. *Chemical Physics Letters*, 89:399–404, 1982.

[4] Marko Balabanovic and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[5] Justin Basilico and Thomas Hofmann. Unifying collaborative and content-based filtering. In Carla E. Brodley, editor, *ICML*. ACM, 2004.

[6] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: using social and content-based information in recommendation. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 714–720, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[7] Jonathan Baxter. A bayesian/information theoretic model of bias learning. In *COLT '96: Proceedings of the ninth annual conference on Computational learning theory*, pages 77–88, New York, NY, USA, 1996. ACM Press.

[8] Jonathan Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine Learning*, 28(1):7–39, 1997.

[9] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[10] S. Ben-David and R. Schuller. Exploiting task relatedness for multiple task learning. In *COLT '03: Proceedings of the Sixteenth Annual Conference on Learning Theory*, 2003.

[11] J. Bicerano. *Prediction of Polymer Properties*. Marcel Dekker, New York, US, 2nd revision edition, 1996.

[12] Christopher Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, London, UK, 1995.

[13] H.J. Böhm and G. Schneider, editors. *Virtual Screening for Bioactive Molecules*, volume 10 of *Methods and Principles in Medicinal Chemistry*. WILEY-VCH, Weinheim, Germany, 2000.

[14] John Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, San Francisco, CA, 1998. Morgan Kaufmann Publishers.

[15] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[16] Rich Caruana. *Multitask Learning*. PhD thesis, Carnegie Mellon University, 1997. Adviser-Tom Mitchell and Herb Simon.

[17] Chemical Computing Group. MOE. `www.chemcomp.com`.

[18] Wei Chu and Zoubin Ghahramani. Gaussian processes for ordinal regression. *J. Mach. Learn. Res.*, 6:1019–1041, 2005.

[19] G. F. Cooper, C. F. Aliferis, R. Ambrosino, J. Aronis, B. G. Buchanan, R. Caruana, M. J. Fine, C. Glymour, G. Gordon, B. H. Hanusa, J. E. Janosky, C. Meek, T. Mitchell, T. Richardson, , and P. Spirtes. An evaluation of machine learning methods for predicting pneumonia mortality. *Artificial Intelligence in Medicine*, 9:107–138, 1997.

[20] K. Crammer and Y. Singer. Pranking with ranking. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[21] Ian Davis and Anthony (Tony) Stentz. Sensor fusion for autonomous outdoor navigation using neural networks. In *Proceedings 1995 IEEE/RSJ International Conference On Intelligent Robotic Systems (IROS '95)*, volume 3, pages 338 – 343, August 1995.

[22] Z. Deng, C. Chuaqui, and J. Singh. Structural interaction fingerprint (sift): A novel method for analyzing three-dimensional protein-ligand binding interactions. *Journal of Medicinal Chemistry*, 47:337–344, 2004.

[23] J. Devillers. *Neural Networks in QSAR and Drug Design.* Academic Press Inc., Orlando, FL, USA, 1996.

[24] J. Devillers. *Genetic Algorithms in Molecular Modeling.* Academic Press, 1999.

[25] J.L. Durant, B.A. Leland, D.R. Henry, and J.G. Nourse. Reoptimization of mdl keys for use in drug discovery. *Journal of Chemical Information and Computer Science*, 42:1273–1280, 2002.

[26] Dumitru Erhan, Pierre-Jean L'Heureux, Shi-Yi Yue, and Yoshua Bengio. Collaborative filtering on a family of biological targets, 2005. Poster Presentation at the 230th ACS Meeting in Washington, DC.

[27] Dumitru Erhan, Pierre-Jean L'Heureux, Shi-Yi Yue, and Yoshua Bengio. Collaborative filtering on a family of biological targets. *Journal of Chemical Information and Modeling*, 46(2):626–635, 2006.

[28] T. Evgeniou, C. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6:615–637, 2005.

[29] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international con-*

ference on Knowledge discovery and data mining, pages 109–117, New York, NY, USA, 2004. ACM Press.

[30] Yoav Freund, Raj D. Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 170–178, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[31] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 209–217, New York, NY, USA, 1998. ACM Press.

[32] J. Ghosn and Y. Bengio. Bias learning, knowledge sharing. *IEEE Transactions on Neural Networks*, 14(4):748–765, Jul 2003.

[33] Joumana Ghosn and Yoshua Bengio. Multi-task learning for stock selection. In Michael Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems 9 (NIPS '96)*, pages 946–952. MIT Press, 1996.

[34] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[35] L.H. Hall and L.B. Kier. *The Molecular Connectivity Chi Indices and Kappa Shape Indices in Structure-Property Modeling*, volume 2 of *Reviews in Computational Chemistry*, chapter 9, pages 367–422. VCH Publishers, Weinheim, Germany, 1991.

[36] L.H. Hall and L.B. Kier. Electrotopological state indices for atom types: A novel combination of electronic, topological, and valence state information. *Journal of Chemical Information and Computer Science*, 35:1039–1045, 1995.

[37] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, April 1982.

[38] C. Hansch and A. Leo. *Exploring QSAR: Fundamentals and Applications in Chemistry and Biology*. ACS Professional Reference Book, 1995.

[39] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *J. Mach. Learn. Res.*, 1:49–75, 2001.

[40] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[41] Thomas Hofmann and Jan Puzicha. Latent class models for collaborative filtering. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 688–693, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[42] Thorsten Joachims. Making large-scale support vector machine learning practical. pages 169–184, 1999.

[43] D. Gupta Ken Goldberg, T. Roeder and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. Technical Report UCB/ERL M00/41, EECS Department, University of California, Berkeley, 2000.

[44] L.B. Kier. Shape indexes of orders one and three from molecular graphs. *Quantitative Structure-Activity Relationships*, 5:1–7, 1986.

[45] R. D. King, J. D. Hirst, and M. J. E. Sternberg. Comparison of artificial intelligence methods for modeling pharmaceutical qsars. *Applied Artificial Intelligence*, 9:213–233, 1995.

[46] H. Kubinyi. Qsar and 3d qsar in drug design part 1: methodology. *Drug Discovery Today*, 2:457–467, 1997.

[47] H. Kubinyi, G. Folkers, and Y.C. Martin, editors. *3D QSAR in Drug Design*, volume 3. Kluwer Academic Publishers, 1998.

[48] Pierre-Jean L'Heureux, Olivier Delalleau, Dumitru Erhan, Yoshua Bengio, and Shi Yi Yue. A neural network application in multi-target QSAR, 2005. Poster Presentation at the 7th International Conference on Chemical Structures, Noordwijkerhout, The Netherlands.

[49] Ming Li and Paul M. B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1993.

[50] Charles X. Ling and Chenghui Li. Data mining for direct marketing: Problems and solutions. In *KDD*, pages 73–79, New York City, New York, USA, 1998. AAAI Press.

[51] Christopher A. Lipinski, Franco Lombardo, Beryl W. Dominy, and Paul J. Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced Drug Delivery Reviews*, 46:3–26, Mar 2001.

[52] Ying Liu. Drug design by machine learning: Ensemble learning for qsar modeling. In *Procedding of The Fourth International Conference on Machine Learning and Applications (ICMLA'05)*, pages 187–193, Los Angelos CA, 2005.

[53] Benjamin Marlin. Modeling user rating profiles for collaborative filtering. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[54] Michinari Momma and Kristin P. Bennett. Sparse kernel partial least squares regression. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *COLT*, volume 2777 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2003.

[55] K. R. Muller, G. Ratsch, S. Sonnenburg, S. Mika, M. Grimm, and N. Heinrich. Classifying 'drug-likeness' with kernel-based learning methods. *Journal of Chemical Information and Modeling*, 45(2):249–253, 2005.

[56] T.I. Oprea. Property distribution of drug-related chemical databases. *Journal of Computer-Aided Molecular Design*, 14:251–264, 2000.

[57] David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 473–480, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[58] M. Petitjean. Applications of the radius-diameter diagram to the classification of topological and geometrical shapes of chemical compounds. *Journal of Chemical Information and Computer Science*, 32:331–337, 1992.

[59] M. Randic. On molecular identification numbers. *Journal of Chemical Information and Computer Science*, 24:164–175, 1984.

[60] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, New York, NY, USA, 1994. ACM Press.

[61] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[62] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. pages 318–362, 1986.

[63] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA, 2001. ACM Press.

[64] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260, New York, NY, USA, 2002. ACM Press.

[65] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[66] R. Smith, C. Hansch, and M. Ames. Selection of a reference partitioning system for drug design work. *Journal of Pharmaceutical Science*, 64:599–606, 1975.

[67] Sebastian Thrun. Lifelong learning: A case study. Technical Report CMU-CS-95-208, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1995.

[68] Volker Tresp and Kai Yu. An introduction to nonparametric hierarchical bayesian modelling with a focus on multi-agent learning. In Roderick Murray-Smith and Robert Shorten, editors, *European Summer School on Multi-AgentControl*, volume 3355 of *Lecture Notes in Computer Science*, pages 290–312. Springer, 2003.

[69] L. H. Ungar and D. P. Foster. Clustering methods for collaborative filtering, 1998. In Workshop on Recommendation Systems at the Fifteenth National Conference on Artificial Intelligence.

[70] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, September 1998.

[71] M. K. Warmuth, J. Liao, G. Ratsch, M. Mathieson, S. Putta, and C. Lemmen. Active learning with support vector machines in the drug discovery process. *Journal of Chemical Information and Modeling*, 43(2):667–673, 2003.

[72] S. Wold, A. Ruhe, H. Wold, and W. J. Dunn. The collinearity problem in linear regression. the partial least squares (PLS) approach to generalized inverses. *SIAM Journal on Scientific Computing*, 5:735–743, 1984.

[73] J. Zupan and J. Gasteiger. *Neural Networks in Chemistry and Drug Design*. Wiley-VCH, Weinheim, Germany, 2nd edition, 1999.